

A Primer for Writing PDE Solvers with Overture

William D. Henshaw¹

Centre for Applied Scientific Computing
 Lawrence Livermore National Laboratory
 Livermore, CA, 94551
 henshaw@llnl.gov
<http://www.llnl.gov/casc/people/henshaw>
<http://www.llnl.gov/casc/Overture> August 7, 2002 UCRL-MA-132233

Abstract: We describe how to write C++ programs to solve partial differential equations on a single curvilinear grid or on a collection of curvilinear grids that form an overlapping grid. We use classes from the Overture framework to represent grids and grid functions, and to perform operations on the grid functions. Overture makes extensive use of the serial/parallel array class library A++/P++ to write efficient and portable serial or parallel code.

Contents

1	Introduction	3
2	Getting Started with MappedGrid's	6
2.1	mappedGridExample1: Mapping's, MappedGrid's, MappedGridFunction's	6
2.2	mappedGridExample2: Solve a PDE on an Annulus using operators	8
2.3	mappedGridExample3: Solve a PDE on an Annulus, explicit BC's, use NameList input	10
2.4	mappedGridExample3CC: Solve a PDE on an Annulus with Finite Volume Operators, explicit BC's, use NameList input	13
2.5	mappedGridExample4: Write your own Mapping and test it	15
2.6	mappedGridExample5: Generate exact solutions in the Twilight Zone	16
2.7	mappedGridExample6: Time Step determination	19
2.8	Example: Steady state, linearized, incompressible Navier-Stokes Equations	23
3	Getting Started with CompositeGrid's	24
3.1	Example 1: CompositeGrid's and MappedGrid's	24
3.2	Example 2: grid functions	27
3.3	Example 3: interpolation	30
3.4	Example 4: show files	31
3.5	Example 5: Differentiating grid functions	33
3.6	Example 6: Solving a simple PDE using Differential and Boundary operators	35
3.7	Example 7: Solving Poisson's equation with Oges	38
3.8	Example 8: Interactive plotting with PlotStuff	40
3.9	Example 9: Saving and Reading a Restart file	42
3.10	Example 10: Efficient computation of derivatives for PDE solvers	44
3.11	Example: 2D Wave equation (optimised for performance and memory usage)	47

¹This work was partially supported by grant N00014-95-F-0067 from the Office of Naval Research

3.12 Example: Moving overlapping grids	49
3.13 Example: Adaptive Grids	52
3.14 Example: Time dependent adaptive mesh refinement solver	54
3.15 Example: Multigrid Overlapping Grids	55
3.16 Example: Solving elliptic problems on each multigrid level	57
3.17 Example: Calling a fortran function for each component grid.	61
4 Single versus double precision	63
5 Makefile's and .cshrc files	64
5.1 Sun Workstations	64
5.2 SGI Workstations	64
5.3 Pentium with Linux	65
6 Variables contained in a MappedGrid	66
7 Variables contained in a CompositeGrid	69

1 Introduction

This is a primer for writing C++ codes to solve partial differential equations (PDEs) within the Overture framework. The first set of examples describe how to use Overture to solve problems on a single curvilinear grid. Overture classes are used to represent the grid, to represent the solution on the grid, to operate on the solution (differentiate, for example) and to plot the results.

The second set of examples show how to solve problems on overlapping grids. These examples are very similar to the single grid case.

The reader is assumed to be familiar with the A++ (parallel) array class library [17]. For the reader wanting to solve problems on overlapping grids, some familiarity with the concept of an overlapping grid is assumed.

In this primer we will introduce and show how to use the following classes

- **Mapping** : A transformation that can be used to represent a curvilinear domain such as a square, annulus, sphere or other more complicated region.
- **MappedGrid** : A logically rectangular grid that is a discrete version of a Mapping. A MappedGrid contains grid point coordinates as well as information such as boundary conditions, periodicity, singularities etc.
- **realMappedGridFunction** : A grid function that holds a solution (such as pressure or velocity) on a MappedGrid; this is a glorified A++ array.
- **MappedGridOperators** : the class used with grid functions to define spatial derivatives and to apply boundary conditions.
- **PlotStuff** The class used to interactively plot Overture objects.
- **Ogshow** : A class for saving solutions and other information in a “show file”. A show file can be read by plotStuff (in the Overture/bin directory) to plot solutions.
- **Oges** : The equation solver class that can be used to solve systems of boundary value problems such as Poisson’s equation.
- **CompositeGrid** : An “overlapping composite grid”. Each component grid of a CompositeGrid is a MappedGrid. The grid generator Ogen, for example, can generate a CompositeGrid.
- **realCompositeGridFunction** : A grid function that holds a solution (such as the pressure or velocity) on a Composite-Grid.
- **CompositeGridOperators** : classes used with grid collection functions to define spatial derivatives and to apply boundary conditions.
- **Ogen** : The overlapping grid generator that can be used in a moving grid computation to regenerate an overlapping grid when one or more of the component grids change. The grid generator can also be run interactively to create an overlapping grid. See the documentation elsewhere.

These classes are collectively known as “Overture”. “Overture” is an acronym that has absolutely no meaning.

Documentation can be found on the Overture home page, <http://www.llnl.gov/casc/Overture>, and includes the following documents that may be of interest

- A++ Quick Reference Card : `A++P++/DOCS/Quick_Reference_Card.tex`
- A primer for Overture[11].
- Grid and grid function documentation[5].
- Finite difference operators and boundary conditions[4].
- Finite volume operators [1].
- Mapping class documentation [6].
- Show file documentation [9].
- Interactive plotting[10].
- Oges “Equation Solver” documentation [8].

- Interactive grid generation documentation [7].
- The other stuff documentation[12].
- The OverBlown Navier-Stokes flow solver [14][13].

Figure 1 gives an overview of the classes that make up Overture.

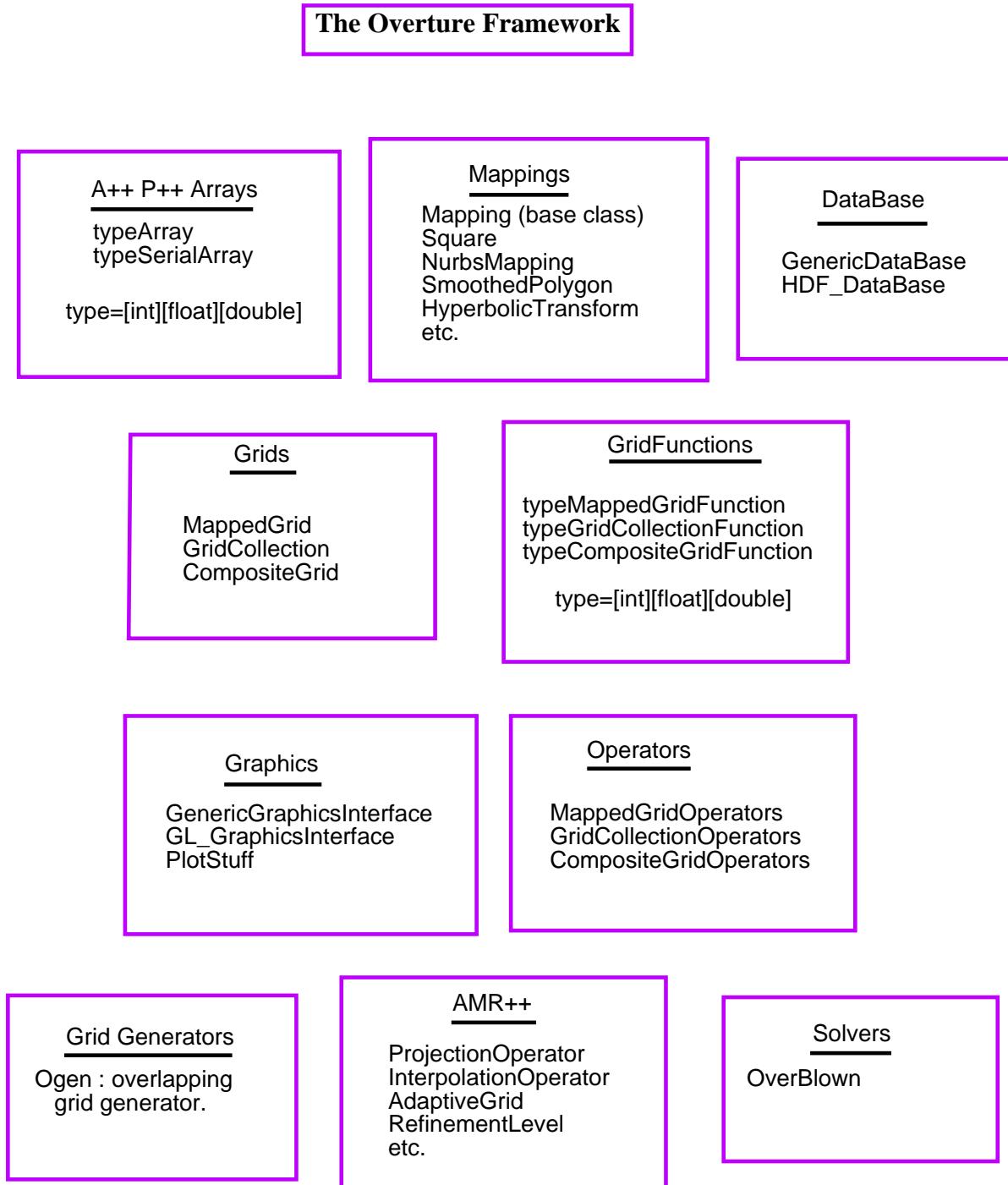


Figure 1: An overview of the Overture classes

2 Getting Started with MappedGrid's

2.1 mappedGridExample1: Mapping's, MappedGrid's, MappedGridFunction's

In this first example we introduce the classes that are used to represent domains (Mapping), grids (MappedGrid) and grid functions (realMappedGridFunction) (a grid function is the Overture name for a solution variable or a field variable).

In this example the SquareMapping is created to define the simple domain that is a square. A MappedGrid is created from the SquareMapping, the MappedGrid is a discrete version of the Mapping since it has a specified number of grid points. To define a solution variable on the grid we build a realMappedGridFunction. This is an A++ array that is automatically dimensioned to the size of the grid. The grid function is assigned values using array operations (instead of looping over all the grid points, which could also be done).

The MappedGridOperator class is used to differentiate the grid function and the PlotStuff class is used to interactively display the grid and the grid function.

Notes when you run the code:

- After some information is printed, a PlotStuff window should appear.
- Use the right mouse button to show the popup menu.
- After choosing the `contour` menu item, a contour plot will appear and you will be in the contour plotter with the contour popup menu. The contour is actually a surface which can be seen by rotating the view by clicking, with the left mouse button on one of the `x-r`, `x+r`, `y-r`, ... buttons.
- Now choose `erase` and `exit` to return to the previous menu.
- Use the left mouse button to look at the `file` pull-down menu (in the upper left corner). Here is the command to save a postscript file. Answers to commands, such as the file name for the postscript file are typed on the Command: line at the bottom. Any current menu item may also be optionally typed on the command line (one need only type at least as many letters in the name as to distinguish it).

(file Overture/primer/mappedGridExample1.C)

```

1 #include "Overture.h"
2 #include "PlotStuff.h"
3 #include "SquareMapping.h"
4 #include "MappedGridOperators.h"
5
6 int
7 main(int argc, char *argv[])
8 {
9     Overture::start(argc,argv); // initialize Overture
10
11    printf(" ----- \n");
12    printf(" Demonstrate mappings, grids, gridfunctions, operators and plotting \n");
13    printf(" ----- \n");
14
15    SquareMapping square(0.,1.,0.,1.); // Make a mapping, unit square
16    square.setGridDimensions(axis1,11); // axis1==0, set no. of grid points
17    square.setGridDimensions(axis2,11); // axis2==1, set no. of grid points
18    MappedGrid mg(square); // MappedGrid for a square
19    // * mg.changeToAllCellCentered(); // make a cell centered grid
20    mg.update(); // This will generate default geometry arrays (e.g. vertex)
21
22    Range all; // a null Range is used as a place-holder below for the coordinates
23    realMappedGridFunction u(mg,all,all,all,2); // create a grid function with 2 components: u(0:10,0:10,0:0,0:1)
24    u.setName("Velocity Stuff"); // give names to grid function ...
25    u.setName("u Stuff",0); // ...and components
26    u.setName("v Stuff",1);
27    Index I1,I2,I3;
28
29    // mg.dimension()(2,3) : start/end index values for all points on the grid, including ghost-points
30    getIndex(mg.dimension(),I1,I2,I3); // assign I1,I2,I3 from dimension
31    u(I1,I2,I3,0)=sin(Pi*mg.vertex()(I1,I2,I3,axis1)) // component 0 : sin(pi*x)*cos(pi*y)
32    *cos(Pi*mg.vertex()(I1,I2,I3,axis2));
33    u(I1,I2,I3,1)=cos(Pi*mg.vertex()(I1,I2,I3,axis1)) // component 1 : cos(pi*x)*sin(pi*y)
34    *sin(Pi*mg.vertex()(I1,I2,I3,axis2));
35

```

```

36     MappedGridOperators op(mg);                                // operators
37     u.setOperators(op);                                     // associate with a grid function
38     u.x().display("here is u.x");                         // x derivative
39     u.x(all,all,all,0).display("here is u.x(all,all,all,0)"); // x derivative of component 0
40     u.x(all,all,all,1).display("here is u.x(all,all,all,1)"); // x derivative of component 1
41
42     getIndex(mg.gridIndexRange(),I1,I2,I3);                // interior and boundary points
43     // compute the error in component 0 of u.x, the notation u.x(I1,I2,I3,0) means only evaluate
44     // the derivative for component 0 and at the points (I1,I2,I3) (done for efficiency only)
45     real error = max(abs( u.x(I1,I2,I3,0)(I1,I2,I3,0)-
46                           Pi*cos(Pi*mg.vertex()(I1,I2,I3,axis1))*cos(Pi*mg.vertex()(I1,I2,I3, axis2)) ) );
47     cout << "Maximum error in component 0 of u.x = " << error << endl;
48
49     bool openGraphicsWindow=TRUE;
50     PlotStuff ps(openGraphicsWindow,"mappedGridExample1");      // create a PlotStuff object
51     PlotStuffParameters psp;                                    // This object is used to change plotting parameters
52
53     aString answer;
54     aString menu[] = {                                         // Make some menu items
55         "!mappedGridExample1",                                // title
56         "contour",
57         "stream lines",
58         "grid",
59         "read command file",
60         "save command file",
61         "erase",
62         "help",
63         "exit",
64         "" };                                              // empty string denotes the end of the menu
65     for(;;)
66     {
67         ps.getMenuItem(menu,answer);                         // put up a menu and wait for a response
68         if( answer=="contour" )
69         {
70             psp.set(GI_TOP_LABEL,"My Contour Plot"); // set title
71             PlotIt::contour(ps,u,psp);                  // contour/surface plots
72         }
73         else if( answer=="grid" )
74             PlotIt::plot(ps,mg,psp);                      // plot the grid
75         else if( answer=="stream lines" )
76             PlotIt::streamLines(ps,u);                   // streamlines
77         else if( answer=="read command file" )
78             ps.readCommandFile();
79         else if( answer=="save command file" )
80             ps.saveCommandFile();
81         else if( answer=="erase" )
82             ps.erase();
83         else if( answer=="help" )
84             helpOverture("PR","mappedGridExample1"); // open web page at documentation
85         else if( answer=="exit" )
86             break;
87     }
88
89     Overture::finish();
90     return 0;
91 }
92

```

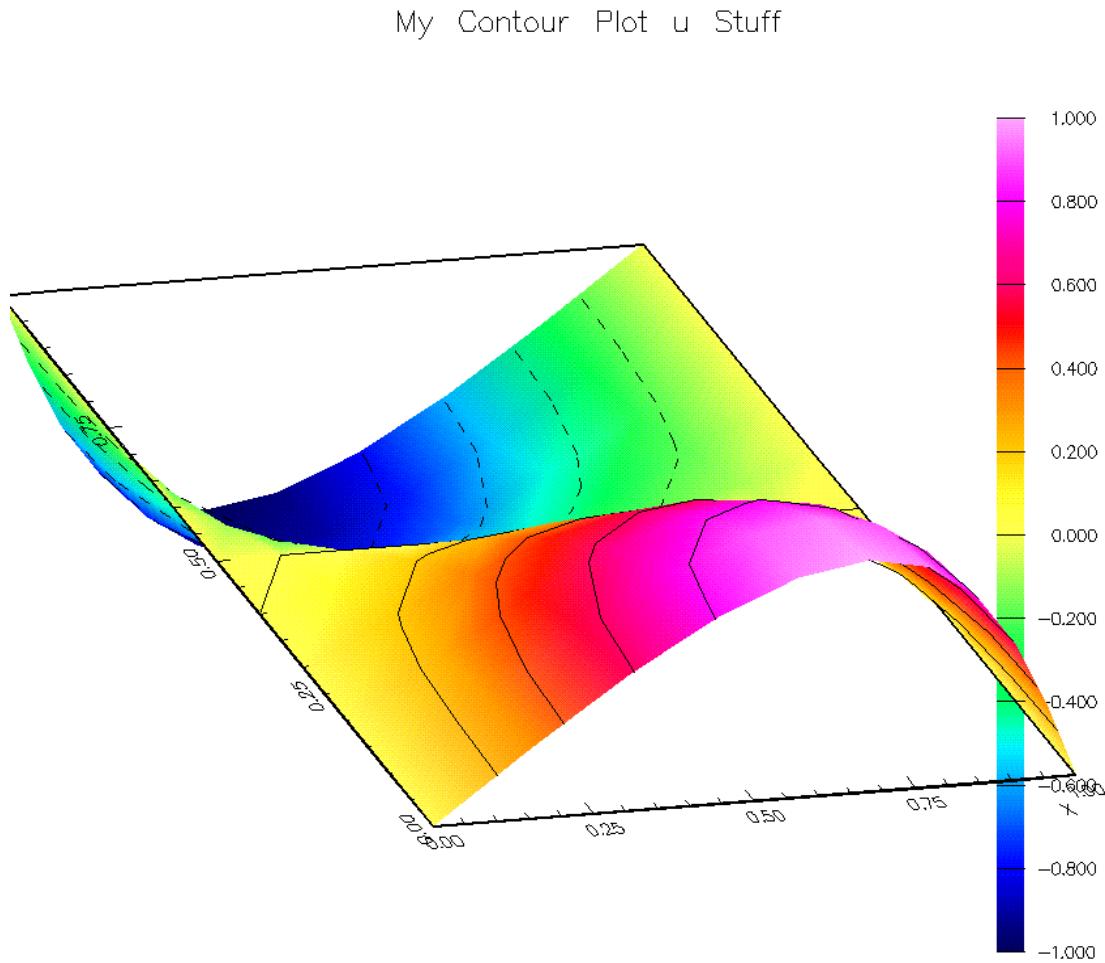


Figure 2: Results from running mappedGridExample1 (contour option, rotated)

2.2 mappedGridExample2: Solve a PDE on an Annulus using operators

In this second example we put together all the classes that were introduced in the first example to solve a simple PDE, a convection diffusion equation:

$$\begin{aligned}
 & u_t + au_x + bu_y = \nu(u_{xx} + u_{yy}), \\
 & u(x, y, 0) = 1 \quad (\text{initial conditions}), \\
 & u(x, y, t) = 0 \quad \text{on the boundary}.
 \end{aligned}$$

An `AnnulusMapping` is used to create an annulus on which to solve the PDE.

The `MappedGridOperator` class is used to apply boundary conditions, in this case a simple Dirichlet boundary condition (i.e. $u = 0$ given on the boundary). Many other boundary conditions are available.

Notes when you run the code:

- The solution will be plotted every 10 steps.
- Choose the `erase` and `exit` menu item from the popup menu to continue another 10 steps.

- you can rotate the view (left mouse button click on x-r a couple of times) to see the solution surface.

(file Overture/primer/mappedGridExample2.C)

```

1 #include "Overture.h"
2 #include "PlotStuff.h"
3 #include "AnnulusMapping.h"
4 #include "MappedGridOperators.h"
5
6 int
7 main(int argc, char *argv[])
8 {
9     Overture::start(argc,argv); // initialize Overture
10
11    printf(" ----- \n");
12    printf(" Solve a convection-diffusion equation on an annulus \n");
13    printf(" Use operators to compute derivatives and apply boundary conditions \n");
14    printf(" Interactively plot results \n");
15    printf(" ----- \n");
16
17    AnnulusMapping annulus;
18    annulus.setGridDimensions(axis1,41);           // axis1==0, set no. of grid points
19    annulus.setGridDimensions(axis2,13);           // axis2==1, set no. of grid points
20    MappedGrid mg(annulus);                      // MappedGrid for a square
21    mg.update();                                // create default variables
22
23    Range all;
24    realMappedGridFunction u(mg);
25    u.setName("Solution");                      // give names to grid function ...
26    u.setName("u",0);                           // ...and components
27
28    Index I1,I2,I3;
29    // The A++ array mg.dimension()(2,3) holds index bounds on all points on the grid, including ghost-points
30    getIndex(mg.dimension(),I1,I2,I3);          // assign I1,I2,I3 from dimension
31    u(I1,I2,I3)=1.;                          // initial conditions
32
33    MappedGridOperators op(mg);                // operators
34    u.setOperators(op);                        // associate with a grid function
35
36    PlotStuff ps(TRUE,"mappedGridExample2");    // create a PlotStuff object
37    PlotStuffParameters psp;                   // This object is used to change plotting parameters
38    char buffer[80];
39
40    real t=0, dt=.005, a=1., b=1., nu=.1;
41    for( int step=0; step<100; step++ )
42    {
43        if( step % 10 == 0 )
44        { // plot contours every 10 steps
45            ps.erase();
46            psp.set(GI_TOP_LABEL,sprintf(buffer,"Solution at time t=%e",t)); // set title
47            PlotIt::contour(ps, u,psp );
48        }
49
50        u+=dt*( (-a)*u.x() +(-b)*u.y() +nu*(u.xx() +u.yy()) ); // ***** forward Euler time step *****
51        t+=dt;
52        // apply Boundary conditions
53        int component=0;
54        u.applyBoundaryCondition(component,BCTypes::dirichlet,BCTypes::allBoundaries,0.); // set u=0.
55        // fix up corners, periodic update:
56        u.finishBoundaryConditions();
57    }
58
59    Overture::finish();
60    return 0;
61 }
62

```

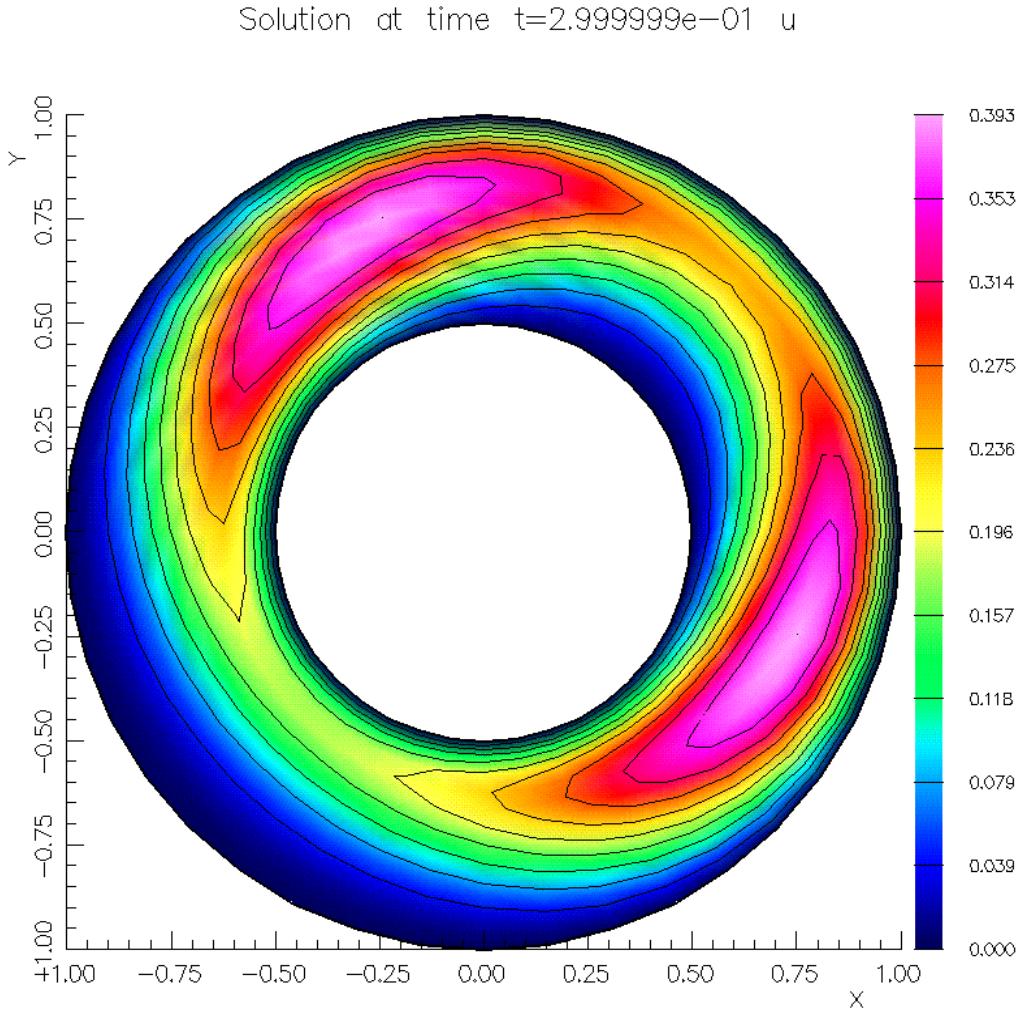


Figure 3: Results from running mappedGridExample2

2.3 mappedGridExample3: Solve a PDE on an Annulus, explicit BC's, use NameList input

In this example we make some minor changes to the previous PDE solver. The `NameList` class is used to interactively enter parameter values by name. This is useful if there are many parameters and only a few need to be changed. The `NameList` class is described in the `PlotStuff` documentation.

Boundary conditions are assigned explicitly just to demonstrate how this is done. Using the operators to apply boundary conditions is easier but there may be cases when the predefined boundary conditions don't do what you want to do.

Notes when you run the code:

- When prompted for changes to the parameters you can type “`numberOfTimeSteps=100`” (or “`nts=100`”) to change the parameter `numberOfTimeSteps`. Type “`exit`” to continue.
- A contour plot of the initial conditions should appear. Choose “`exit`” from the popup menu (right mouse button) and the solution will be automatically displayed every 5 time steps (“movie mode”).

(file `Overture/primer/mappedGridExample3.C`)

```

1 #include "Overture.h"
2 #include "PlotStuff.h"
```

```

3 #include "AnnulusMapping.h"
4 #include "MappedGridOperators.h"
5 #include "NameList.h"
6
7 int
8 main(int argc, char *argv[])
9 {
10     Overture::start(argc,argv); // initialize Overture
11
12     printf(" ----- \n");
13     printf(" Solve a convection-diffusion equation on an annulus \n");
14     printf(" Input parameters with the NameList class \n");
15     printf(" Show how to apply boundary conditions explicitly \n");
16     printf(" ----- \n");
17
18     // Set default values for parameters. These can be optionally changed below
19     int numberofTimeSteps=1000;
20     real dt=.01;
21     IntegerArray bc(2,3); bc=1;
22     NameList nl; // The NameList object allows one to read in values by name
23     aString name(80),answer(80);
24     printf(
25         " Parameters for Example 3: \n"
26         " ----- \n"
27         " name type default \n"
28         "numberofTimeSteps (nts=) (int) %i \n"
29         "time step (dt=) (real) %f \n"
30         "boundary conditions (bc(side,axis)=) (IntegerArray) \n",
31         numberofTimeSteps,dt);
32
33 // =====Loop for changing parameters=====
34 for( ; ; )
35 {
36     cout << "Enter changes to variables, exit to continue" << endl;
37     getLine(answer);
38     if( answer=="exit" ) break;
39     nl.getVariableName( answer, name ); // parse the answer
40     if( name== "numberofTimeSteps" || name=="nts" )
41     {
42         numberofTimeSteps=nl.intValue(answer);
43         cout << "numberofTimeSteps = " << numberofTimeSteps << endl;
44     }
45     else if( name== "dt" )
46         dt=nl.realValue(answer);
47     else if( name== "bc" )
48         nl.getIntArray( answer,bc );
49     else
50         cout << "unknown response: [ " << name << " ]" << endl;
51 }
52
53 Mapping *mapping; // keep a pointer to a mapping
54 mapping = new AnnulusMapping(); // create an Annulus
55 mapping->setGridDimensions(axis1,41); // axis1==0, set no. of grid points
56 mapping->setGridDimensions(axis2,13); // axis2==1, set no. of grid points
57 MappedGrid mg(*mapping); // MappedGrid for a square
58 mg.update(); // create default variables
59
60 Range all;
61 realMappedGridFunction u;
62 u.updateToMatchGrid(mg,all,all,all,1); // define after declaration (like resize)
63 u.setName("Solution"); // give names to grid function ...
64 u.setName("u",0); // ...and components
65
66 Index I1,I2,I3, Ib1,Ib2,Ib3;
67 // The A++ array mg.dimension()(2,3) holds index bounds on all points on the grid, including ghost-points
68 getIndex(mg.dimension(),I1,I2,I3); // assign I1,I2,I3 from dimension
69 u(I1,I2,I3)=1.; // initial conditions
70
71 MappedGridOperators op(mg); // operators
72 u.setOperators(op); // associate with a grid function
73
74 PlotStuff ps(TRUE,"mappedGridExample3"); // create a PlotStuff object

```

```

75 PlotStuffParameters psp;                                // This object is used to change plotting parameters
76 char buffer[80];
77
78 real t=0, a=1., b=1., nu=.05;
79 for( int step=0; step<numberOfTimeSteps; step++ )
80 {
81     if( step % 5 == 0 )
82     {
83         psp.set(GI_TOP_LABEL,sprintf(buffer,"Solution at time t=%e",t)); // set title
84         ps.erase();
85         PlotIt::contour(ps, u,psp );
86         psp.set(GI_PLOT_THE_OBJECT_AND_EXIT,TRUE);      // set this to run in "movie" mode (after first plot)
87         ps.redraw(TRUE);
88     }
89     u+=dt*( (-a)*u.x() +(-b)*u.y() +nu*(u.xx() +u.yy()) );
90     t+=dt;
91     // apply Boundary conditions explicitly (just to demonstrate, it is easier to use the operators)
92     for( int axis=0; axis<mg.numberOfDimensions(); axis++ )
93         for( int side=Start; side<=End; side++ )
94         { // only assign BC's on sides with a positive boundary condition:
95             if( mg.boundaryCondition()(side,axis) > 0 )
96                 { // fill in boundary values
97                     getBoundaryIndex(mg.gridIndexRange(),side,axis,Ib1,Ib2,Ib3);
98                     u(Ib1,Ib2,Ib3)=0.;
99                 }
100         }
101     u.periodicUpdate(); // swap periodic edges
102 }
103
104 Overture::finish();
105 return 0;
106 }
107

```

2.4 mappedGridExample3CC: Solve a PDE on an Annulus with Finite Volume Operators, explicit BC's, use NameList input

We repeat the last example but now using the finite volume operators on a cell centered grid.
 (file Overture/primer/mappedGridExample3CC.C)

```

1  #include "Overture.h"
2  #include "PlotStuff.h"
3  #include "AnnulusMapping.h"
4  #include "MappedGridFiniteVolumeOperators.h"
5  #include "NameList.h"
6
7  int
8  main()
9  {
10    ios::sync_with_stdio();      // Synchronize C++ and C I/O subsystems
11    Index::setBoundsCheck(on); // Turn on A++ array bounds checking
12
13    printf(" ----- \n");
14    printf(" Solve a convection-diffusion equation on an annulus \n");
15    printf(" Input parameters with the NameList class \n");
16    printf(" Show how to apply boundary conditions explicitly \n");
17    printf(" ----- \n");
18
19    // Set default values for parameters. These can be optionally changed below
20    int numberoftimesteps=1000;
21    real dt=.01;
22    IntegerArray bc(2,3); bc=1;
23    NameList nl;           // The NameList object allows one to read in values by name
24    aString name(80),answer(80);
25    printf(
26      " Parameters for Example 3: \n"
27      " ----- \n"
28      " name                      type   default \n"
29      " numberoftimesteps (nts=)    (int)   %i   \n"
30      " time step (dt=)           (real)  %f   \n"
31      " boundary conditions (bc(side,axis)=) (IntegerArray) \n",
32      numberoftimesteps,dt);
33
34 // =====Loop for changing parameters=====
35 for( ; )
36 {
37   cout << "Enter changes to variables, exit to continue" << endl;
38   getLine(answer);
39   if( answer=="exit" ) break;
40   nl.getVariableName( answer, name ); // parse the answer
41   if( name== "numberoftimesteps" || name=="nts" )
42     numberoftimesteps=nl.intValue(answer);
43   else if( name== "dt" )
44     dt=nl.realValue(answer);
45   else if( name== "bc" )
46     nl.getIntArray( answer,bc );
47   else
48     cout << "unknown response: [ " << name << " ]" << endl;
49 }
50
51 Mapping *mapping;                                // keep a pointer to a mapping
52 mapping = new AnnulusMapping();                  // create an Annulus
53 mapping->setGridDimensions(axis1,41);          // axis1=-0, set no. of grid points
54 mapping->setGridDimensions(axis2,13);          // axis2=1, set no. of grid points
55 MappedGrid mg(*mapping);                        // MappedGrid for a square
56 mg.changeToAllCellCentered();                   // create default variables
57 mg.update();
58
59 Range all;
60 realMappedGridFunction u;                       // define after declaration (like resize)
61 u.updateToMatchGrid(mg,all,all,all,1);          // give names to grid function ...
62 u.setName("Solution");                         // ...and components
63 u.setName("u",0);
64
65 Index I1,I2,I3, Ib1,Ib2,Ib3;
66 // The A++ array mg.dimension()(2,3) holds index bounds on all points on the grid, including ghost-points

```

```

67     getIndex(mg.dimension(),I1,I2,I3);           // assign I1,I2,I3 from dimension
68     u(I1,I2,I3)=1.;                           // initial conditions
69
70     MappedGridFiniteVolumeOperators op(mg);      // operators
71     u.setOperators(op);                         // associate with a grid function
72
73     PlotStuff ps(TRUE,"mappedGridExample3");    // create a PlotStuff object
74     PlotStuffParameters psp;                    // This object is used to change plotting parameters
75     char buffer[80];
76
77     real t=0, a=1., b=1., nu=.05;
78     for( int step=0; step<numberOfTimeSteps; step++ )
79     {
80         if( step % 5 == 0 )
81         {
82             psp.set(GI_TOP_LABEL,sprintf(buffer,"Solution at time t=%e",t)); // set title
83             ps.erase();
84             PlotIt::contour(ps, u,psp );
85             psp.set(GI_PLOT_THE_OBJECT_AND_EXIT,TRUE);      // set this to run in "movie" mode (after first plot)
86             ps.redraw(TRUE);
87         }
88         u+=dt*( (-a)*u.x() +(-b)*u.y() +nu*(u.laplacian()) );
89         t+=dt;
90         // apply Boundary conditions explicitly (just to demonstrate, it is easier to use the operators)
91         for( int axis=0; axis<mg.numberOfDimensions(); axis++ )
92             for( int side=Start; side<=End; side++ )
93             { // only assign BC's on sides with a positive boundary condition:
94                 if( mg.boundaryCondition()(side,axis) > 0 )
95                 { // fill in boundary values
96                     getBoundaryIndex(mg.gridIndexRange(),side,axis,Ib1,Ib2,Ib3);
97                     u(Ib1,Ib2,Ib3)=0.; // ***** this is not correct for cell-centred case ***
98                 }
99             }
100            u.periodicUpdate(); // swap periodic edges
101        }
102
103        return 0;
104    }
105

```

2.5 mappedGridExample4: Write your own Mapping and test it

This example can be skipped for first time users. This program can be used to test a new Mapping if you have written one.
 (file Overture/primer/mappedGridExample4.C)

```

1 #include "Overture.h"
2 #include "PlotStuff.h"
3 #include "ChannelMapping.h"
4 #include "MappingInformation.h"
5 #include "HDF_DataBase.h"
6
7 int
8 main(int argc, char *argv[])
9 {
10   Overture::start(argc,argv); // initialize Overture
11
12   printf(" ----- \n");
13   printf(" Test a user-written Mapping, ChannelMapping.{h,C} \n");
14   printf(" Use the checkMapping function to test derivatives, consistency of parameters etc. \n");
15   printf(" Save the mapping to a data base and read back again \n");
16   printf(" ----- \n");
17
18   ChannelMapping channel;
19
20   realArray r(1,2),x(1,2),xr(1,2,2);
21   r=.5;
22   channel.map(r,x,xr);
23   printf(" r=(%f,%f) x=(%f,%f) xr=(%f,%f,%f,%f)\n",r(0,0),r(0,1),x(0,0),x(0,1),
24   xr(0,0,0),xr(0,1,0),xr(0,0,1),xr(0,1,1));
25
26   // this function will check the mapping and it's derivatives etc.
27   channel.checkMapping();
28
29
30   // Make interactive changes to the mapping
31   PlotStuff ps(TRUE,"mappedGridExample4");           // create a PlotStuff object
32   MappingInformation mapInfo;                      // parameters used by map.update
33   mapInfo.graphXInterface=&ps;                     // pass graphics interface
34   channel.update(mapInfo);
35
36   // Save the mapping in a data-base file
37   HDF_DataBase DataBase;
38   cout << "Mount a new database file...\n";
39   DataBase.mount("map.dat","I");                  // Initialize a database file
40
41   channel.put(DataBase,"my-channel");
42   DataBase.unmount();
43
44   // now mount the data-base and read in the mapping
45   cout << "Mount an old data base file and read a mapping from it...\n";
46   DataBase.mount("map.dat","R"); // mount a data base read-only
47   ChannelMapping channel2;
48   channel2.get(DataBase,"my-channel");
49
50
51   r=1.;
52   channel2.map(r,x,xr);
53   printf(" r=(%f,%f) x=(%f,%f) xr=(%f,%f,%f,%f)\n",r(0,0),r(0,1),x(0,0),x(0,1),
54   xr(0,0,0),xr(0,1,0),xr(0,0,1),xr(0,1,1));
55
56   Overture::finish();
57   return 0;
58 }
59

```

2.6 mappedGridExample5: Generate exact solutions in the Twilight Zone

In this example we show how to use the **method of analytic solutions** to generate an exact solution to a PDE (the so-called twilight-zone solution). Given a PDE that we want to solve, such as,

$$\begin{aligned} u_t + au_x + bu_y &= \nu(u_{xx} + u_{yy}) + f(x, y, t), \\ u(x, y, t) &= g(x, y, t) \quad \text{on the boundary}, \end{aligned}$$

we can choose the forcing function $f(x, y, t)$ and boundary condition $g(x, y, t)$ so that the solution is known. Suppose that we want the solution to be the known function $U(x, y, t)$. For example U may be the polynomial

$$U = (1 + x + x^2)(1 + y + y^2)(1 + t + t^2).$$

By choosing

$$\begin{aligned} f(x, y, t) &= U_t + aU_x + bU_y - \nu(U_{xx} + U_{yy}) \\ g(x, y, t) &= U(x, y, t) \end{aligned}$$

the exact solution for $u(x, y, t)$ will then be $U(x, y, t)$.

The classes `OGPolyFunction` and `OGTrigFunction` have been written to define polynomial and trigonometric functions and their derivatives that can be used to define the known function $U(x, y, z, t)$.

The boundary conditions defined in the operators have been written to optionally use this method of analytic solutions. The operators need to be given the appropriate true solution and they need to be told to force the boundary conditions as shown in the example code below.

When using the polynomial exact solution (default) in the example below the errors in the computed solution are ‘zero’ (to roundoff, 2×10^{-7} in IEEE single precision).

When using the trigonometric exact solution (by choosing “f=2”) the errors at time $t = .5$ are 3.9×10^{-3} in IEEE single precision.

(file `Overture/primer/mappedGridExample5.C`)

```

1  #include "Overture.h"
2  #include "PlotStuff.h"
3  #include "SquareMapping.h"
4  #include "AnnulusMapping.h"
5  #include "MappedGridOperators.h"
6  #include "NameList.h"
7  #include "OGTrigFunction.h"
8  #include "OGPolyFunction.h"
9
10 enum forcingTypes
11 {
12     noForcing=0,
13     poly,
14     trig
15 };
16
17 int
18 main(int argc, char *argv[])
19 {
20     Overture::start(argc,argv); // initialize Overture
21
22     printf(" -----\n");
23     printf(" Solve a convection-diffusion equation on an annulus\n");
24     printf(" Use the method of analytic solutions to obtain an exact solution\n");
25     printf(" Use OGPolyFunction and OGTrigFunction to define the exact solution and it's derivatives\n");
26     printf(" -----\n");
27
28     // Set default values for parameters. These can be optionally changed below
29     int numberoftimesteps=100;
30     real dt=.005;
31     IntegerArray bc(2,3); bc=1;
32     IntegerArray gridpoints(3); gridpoints=-1;
33     int mapType=0; // 0=square, 1=annulus
34     forcingTypes forcingoption=poly;
35     int plotoption=TRUE;
36
37     // The NameList object allows one to read in values by name

```

```

38     NameList nl;
39     aString name(80),answer(80);
40     printf(
41         " Parameters for Example 5: \n"
42         " ----- \n"
43         " name                                     type   default \n"
44         " numberofTimeSteps (nts=)                  (int)   %i   \n"
45         " mapType (mt= 0:square, 1=annulus)          (int)   %i   \n"
46         " forcingOption (f= 0:none, 1=poly, 2=trig)    (int)   %i   \n"
47         " plotOption (p = 1:on, 0:off)                 (int)   %i   \n"
48         " time step (dt)                           (real)  %f   \n"
49         " gridPoints(axis) (gp(axis)=no. of grid points) (IntegerArray) \n"
50         " boundary conditions (bc(side,axis)=)        (IntegerArray) \n",
51             numberofTimeSteps, mapType, forcingOption, plotOption, dt);
52
53 // =====Loop for changing parameters=====
54 for( ; ; )
55 {
56     cout << "Enter changes to variables, exit to continue" << endl;
57     getLine(answer);
58     if( answer=="exit" ) break;
59     nl.getVariableName( answer, name ); // parse the answer
60     if( name== "numberofTimeSteps" || name=="nts" )
61         numberofTimeSteps=nl.intValue(answer);
62     else if( name== "dt" )
63         dt=nl.realValue(answer);
64     else if( name== "mapType" || name=="mt" )
65         mapType=nl.intValue(answer);
66     else if( name== "forcingOption" || name=="f" )
67         forcingOption=(forcingTypes)nl.intValue(answer);
68     else if( name== "plotOption" || name=="p" )
69         plotOption=nl.intValue(answer);
70     else if( name== "bc" )
71         nl.getIntArray( answer,bc );
72     else if( name== "gridPoints" || name=="gp" )
73         nl.getIntArray( answer,gridPoints );
74     else
75         cout << "unknown response: [ " << name << " ]" << endl;
76 }
77
78 Mapping *mapping;                                // keep a pointer to a mapping
79 if( mapType==0 )
80 {
81     mapping = new SquareMapping();                // create a Square
82     mapping->setGridDimensions(axis1,11);          // axis1==0, set no. of grid points
83     mapping->setGridDimensions(axis2,11);          // axis2==1, set no. of grid points
84 }
85 else
86 {
87     mapping = new AnnulusMapping();               // create an Annulus
88     mapping->setGridDimensions(axis1,41);          // axis1==0, set no. of grid points
89     mapping->setGridDimensions(axis2,13);          // axis2==1, set no. of grid points
90 }
91 for( int axis=0; axis<mapping->getDomainDimension(); axis++ )
92 {
93     if( gridPoints(axis)>0 )
94         mapping->setGridDimensions(axis,gridPoints(axis));
95 }
96 MappedGrid mg(*mapping);                         // MappedGrid for a square
97 mg.update();                                    // create default variables
98
99 Range all;
100 realMappedGridFunction u(mg);
101 u.setName("Solution");                         // give names to grid function ...
102 u.setName("u",0);                             // ...and components
103
104 OGFunction *exactPointer;
105 if( forcingOption==poly )
106 {
107     int degreeOfSpacePolynomial = 2;
108     int degreeOfTimePolynomial = 1;
109     int nComp = 1;

```

```

110     exactPointer = new OGPolyFunction(degreeOfSpacePolynomial,mg.numberOfDimensions(),nComp,
111                                     degreeOfTimePolynomial);
112 }
113 else if( forcingOption==trig )
114 {
115     real fx=1., fy = 1., fz = 1., ft=1.;           // note that fz is not used in 2D
116     // defines cos(pi*x)*cos(pi*y)*cos(pi*z)*cos(pi*t)
117     exactPointer = new OGTrigFunction(fx, fy, fz, ft);
118 }
119 else if( forcingOption!=0 )
120 {
121     cout << "Unknown forcing option = " << forcingOption << endl;
122     forcingOption=noForcing;
123 }
124 OGFunctor & exact = *exactPointer; // make a reference for readability
125
126 Index I1,I2,I3, Ib1,Ib2,Ib3;
127 // mg.dimension()(2,3) : all points on the grid, including ghost-points
128 getIndex(mg.dimension(),I1,I2,I3);           // assign I1,I2,I3 from dimension
129 realArray & x= mg.vertex();
130 if( forcingOption > 0 )
131     u(I1,I2,I3)=exact(mg,I1,I2,I3,0,0.);
132 else
133     u=1.;
134
135 MappedGridOperators op(mg);                  // operators
136 u.setOperators(op);                         // associate with a grid function
137 // ***** tell the operators to use the method of analytic solutions for BC's *****
138 //      if the forcingOption is greater than 0
139 if( forcingOption>0 )
140 {
141     op.setTwilightZoneFlow(TRUE);
142     op.setTwilightZoneFlowFunction(exact);
143 }
144
145 PlotStuff ps(plotOption,"mappedGridExample5"); // create a PlotStuff object
146 PlotStuffParameters psp;                      // This object is used to change plotting parameters
147 char buffer[80];
148
149 // Index's for boundary and interior points:
150 getIndex(mg.gridIndexRange(),I1,I2,I3);
151 real t=0, a=1., b=1., nu=.1;
152 for( int step=0; step<numberOfTimeSteps; step++ )
153 {
154     if( plotOption && step % 20 == 0 )
155     {
156         sprintf(buffer,"Solution at time t=%e",t);
157         psp.set(GI_TOP_LABEL,buffer); // set title
158         PlotIt::contour(ps, u,psp );
159     }
160
161     u+=dt*( (-a)*u.x() +(-b)*u.y() +nu*(u.xx() +u.yy() ) );
162     if( forcingOption > 0 )
163     { // ***** Here we add on dt[ U_t + a U_x + b U_y - nu( U_xx + U_yy ) ] *****
164         u(I1,I2,I3)+=dt*(exact.t(mg,I1,I2,I3,0,t)
165                           + a*exact.x(mg,I1,I2,I3,0,t) + b*exact.y(mg,I1,I2,I3,0,t)
166                           - nu*( exact.xx(mg,I1,I2,I3,0,t) + exact.yy(mg,I1,I2,I3,0,t) ) );
167     }
168     t+=dt;
169     // apply Boundary conditions, this will set u=exact if forcingOption>0
170     u.applyBoundaryCondition(0,BCTypes::dirichlet,BCTypes::allBoundaries,0.,t);
171     // fix up corners, periodic update:
172     u.finishBoundaryConditions();
173
174     real error = max(abs( u(I1,I2,I3)-exact(mg,I1,I2,I3,0,t)));
175     printf("t=%6.3f error =%e \n",t,error);
176 }
177
178 Overture::finish();
179 return 0;
180 }
181

```

2.7 mappedGridExample6: Time Step determination

In this example we demonstrate how the time step can be accurately determined for a convection diffusion equation

$$u_t + au_x + bu_y = \nu(u_{xx} + u_{yy})$$

The function that determines the time step is in file Overture/primer/getDt.C while the file Overture/primer/-mappedGridExample6.C uses this function. See the document *Time Step Determination for PDEs with Applications to Programs Written with Overture* [15] (available from the Overture home page) for further details.

(file Overture/primer/getDt.C)

```

1 //=====
2 //
3 // This function is used by mappedGridExample6
4 //
5 //=====
6 #include "Overture.h"
7 #include "MappedGridOperators.h"
8
9 real
10 getDt(const real & cfl,
11        const real & a,
12        const real & b,
13        const real & nu,
14        MappedGrid & mg,
15        MappedGridOperators & op,
16        const real alpha0 = -2.,
17        const real beta0 = 1. )
18 //=====
19 // /Description:
20 // Determine the time step for the convection diffusion equation
21 //      u_t + a u_x + b u_y = nu( u_xx + u_yy )
22 // discretized with the mapping method. Scale the maximum allowable time
23 // step by the factor cfl. the stability region is assumed to lie within the
24 // ellipse (x/alpha0)^2 + (y/beta0)^2 = 1
25 // /cfl (input): Scale the time step by this factor (cfl=1 should normally be stable)
26 // /a (input) : coefficient of u_x
27 // /b (input) : coefficient of u_y
28 // /nu (input) : coefficient of (u_xx+u_yy)
29 // /alpha0, beta0 (input) : parameters defining the ellipse for the stability region
30 // =====
31 {
32     real dt;
33     if( mg.isRectangular() )
34     {
35         // ***** rectangular grid *****
36         real dx[3];
37         mg.getDeltaX(dx);
38
39         dt = cfl * pow(
40             pow( fabs(a)*(1./(beta0*dx[0]))+fabs(b)*(1./beta0*dx[1]) , 2.)
41             +pow( nu *(4./(alpha0*dx[0]*dx[0])+4./(alpha0*dx[1]*dx[1])) , 2.)
42             ,-.5);
43     }
44     else
45     {
46         // ***** non-rectangular grid *****
47         mg.update(MappedGrid::THEinverseVertexDerivative); // make sure the jacobian derivatives are built
48         // define an alias:
49         realMappedGridFunction & rx = mg.inverseVertexDerivative();
50         // we need to compute the derivatives of rx:
51         rx.setOperators(op);
52         // Get Index's for the interior+boundary points
53         Index I1,I2,I3;
54         getIndex( mg.indexRange(),I1,I2,I3 );
55
56         realArray a1,b1,nul1,nul2,nu22;
57         // a1 = a*r1.x + b*r1.y + nu ( r1.xx + r1.yy )      b1 = a*r2.x + b*r2.y + nu ( r2.xx + r2.yy )
58         a1 = a*rx(I1,I2,I3,0,0) + b*rx(I1,I2,I3,0,1)
59         - nu*( rx.x(I1,I2,I3,0,0)(I1,I2,I3,0,0) + rx.y(I1,I2,I3,0,1)(I1,I2,I3,0,1) );
60         b1 = a*rx(I1,I2,I3,1,0) + b*rx(I1,I2,I3,1,1)

```

```

61      - nu*( rx.x(I1,I2,I3,1,0)(I1,I2,I3,1,0) + rx.y(I1,I2,I3,1,1)(I1,I2,I3,1,1) );
62 // nul1 = nu*( r1.x*r1.x + r1.y*r1.y )
63 // nul2 = nu*( r1.x*r2.x + r1.y*r2.y )*2
64 // nu22 = nu*( r2.x*r2.x + r2.y*r2.y )
65 null = nu*( rx(I1,I2,I3,0,0)*rx(I1,I2,I3,0,0) + rx(I1,I2,I3,0,1)*rx(I1,I2,I3,0,1) );
66 nul2 = nu*( rx(I1,I2,I3,0,0)*rx(I1,I2,I3,1,0) + rx(I1,I2,I3,0,1)*rx(I1,I2,I3,1,1) )*2.;
67 nu22 = nu*( rx(I1,I2,I3,1,0)*rx(I1,I2,I3,1,0) + rx(I1,I2,I3,1,1)*rx(I1,I2,I3,1,1) );
68
69 // Grid spacings on unit square:
70 real dr1 = mg.gridSpacing()(axis1);
71 real dr2 = mg.gridSpacing()(axis2);
72 dt = cfl * min(
73     pow(
74         pow( abs(a1)*(1./(beta0*dr1))+abs(b1)*(1./beta0*dr2) , 2.)
75         +pow( null * (4./(alpha0*dr1*dr1))
76             +abs(nu12)*(1./(alpha0*dr1*dr2))
77             +nu22 *(4./(alpha0*dr2*dr2)) , 2.)
78         ,-.5)
79    );
80 }
81 return dt;
82 }

(file Overture/primer/mappedGridExample6.C)

1 //=====
2 //
3 // This example shows how to determine the time step for a 2D convection diffusion equation
4 //
5 // Bill Henshaw
6 //=====
7 #include "Overture.h"
8 #include "PlotStuff.h"
9 #include "SquareMapping.h"
10 #include "AnnulusMapping.h"
11 #include "MappedGridOperators.h"
12 #include "NameList.h"
13
14 #define UTRUE(x,y,t) (x)*(1.-(x))*(y)*(1.-(y))*(1.+(t))
15 #define UTRUEX(x,y,t) (1.-2.*(x))*(y)*(1.-(y))*(1.+(t))
16 #define UTRUEY(x,y,t) (x)*(1.-(x))*(1.-2.*(y))*(1.+(t))
17 #define UTRUET(x,y,t) (x)*(1.-(x))*(y)*(1.-(y))
18 #define UTRUEXX(x,y,t) -2.*(y)*(1.-(y))*(1.+(t))
19 #define UTRUEYY(x,y,t) -2.*(x)*(1.-(x))*(1.+(t))
20
21 #define FORCE(x,y,t) UTRUET(x,y,t)+a*UTRUEX(x,y,t)+b*UTRUEY(x,y,t) \
22           -nu*(UTRUEXX(x,y,t)+UTRUEYY(x,y,t))
23
24 real
25 getDt(const real & cfl,
26       const real & a,
27       const real & b,
28       const real & nu,
29       MappedGrid & mg,
30       MappedGridOperators & op,
31       const real alpha0 = -2.,
32       const real beta0 = 1. );
33
34 int
35 main(int argc, char *argv[])
36 {
37   Overture::start(argc,argv); // initialize Overture
38
39   printf(" ----- \n");
40   printf(" Solve a convection-diffusion equation on a square or annulus \n");
41   printf(" Determine the correct time step using the getDt function (getDt.C) \n");
42   printf(" ----- \n");
43
44   // Set default values for parameters. These can be optionally changed below
45   int numberoftimesteps=100;
46   real dt=.005, cfl=.5;
47   IntegerArray bc(2,3); bc=1;
48   IntegerArray gridpoints(3); gridpoints=-1;

```

```

49 int mapType=0; // 0=square, 1=annulus
50 // The NameList object allows one to read in values by name
51 NameList nl;
52 aString name(80),answer(80);
53 printf(
54 " Parameters for Example 3: \n"
55 " ----- \n"
56 " name type default \n"
57 "numberOfTimeSteps (nts=) (int) %i \n"
58 "mapType (mt= 0:square, 1=annulus) (int) %i \n"
59 "cfl (real) %f \n"
60 "time step (dt=) (real) %f \n"
61 "gridPoints(axis) (gp(axis)=no. of grid points) (intArray) \n"
62 "boundary conditions (bc(side,axis)=) (intArray) \n",
63     numberOfTimeSteps,mapType,cfl,dt);
64
65 // =====Loop for changing parameters=====
66 for( ; ; )
67 {
68     cout << "Enter changes to variables, exit to continue" << endl;
69     getLine(answer);
70     if( answer=="exit" ) break;
71     nl.getVariableName( answer, name ); // parse the answer
72     if( name== "numberOfTimeSteps" || name=="nts" )
73         numberOfTimeSteps=nl.intValue(answer);
74     else if( name== "dt" )
75         dt=nl.realValue(answer);
76     else if( name== "cfl" )
77         cfl=nl.realValue(answer);
78     else if( name== "mapType" || name=="mt" )
79         mapType=nl.intValue(answer);
80     else if( name== "bc" )
81         nl.getIntArray( answer,bc );
82     else if( name== "gridPoints" || name=="gp" )
83         nl.getIntArray( answer,gridPoints );
84     else
85         cout << "unknown response: [ " << name << "]" << endl;
86
87 }
88
89 Mapping *mapping; // keep a pointer to a mapping
90 if( mapType==0 )
91 {
92     mapping = new SquareMapping(); // create a Square
93     mapping->setGridDimensions(axis1,11); // axis1==0, set no. of grid points
94     mapping->setGridDimensions(axis2,11); // axis2==1, set no. of grid points
95 }
96 else
97 {
98     mapping = new AnnulusMapping(); // create an Annulus
99     mapping->setGridDimensions(axis1,41); // axis1==0, set no. of grid points
100    mapping->setGridDimensions(axis2,13); // axis2==1, set no. of grid points
101 }
102 for( int axis=0; axis<mapping->getDomainDimension(); axis++ )
103 {
104     if( gridPoints(axis)>0 )
105         mapping->setGridDimensions(axis,gridPoints(axis));
106 }
107 MappedGrid mg(*mapping); // MappedGrid for a square
108 mg.update(); // create default variables
109
110 Range all;
111 realMappedGridFunction u(mg);
112 u.setName("Solution"); // give names to grid function ...
113 u.setName("u",0); // ...and components
114
115 Index I1,I2,I3, Ib1,Ib2,Ib3;
116 // mg.dimension()(2,3) : all points on the grid, including ghost-points
117 getIndex(mg.dimension(),I1,I2,I3); // assign I1,I2,I3 from dimension
118 realArray & x= mg.vertex();
119 u(I1,I2,I3)=UTRUE(x(I1,I2,I3,0),x(I1,I2,I3,1),0.); // initial conditions
120

```

```

121 MappedGridOperators op(mg);           // operators
122 u.setOperators(op);                  // associate with a grid function
123
124 PlotStuff ps(TRUE,"mappedGridExample6"); // create a PlotStuff object
125 PlotStuffParameters psp;             // This object is used to change plotting parameters
126 char buffer[80];
127
128 real t=0, a=1., b=1., nu=.1;
129
130 // getDt needs the inverseVertexDerivative
131 mg.update(MappedGrid::THEinverseVertexDerivative);
132
133 dt = getDt( cfl,a,b,nu,mg,op );
134 cout << " dt from getDt = " << dt << endl;
135
136 int tStep=numberOfTimeSteps/10;
137
138 for( int step=0; step<numberOfTimeSteps; step++ )
139 {
140     if( step % tStep == 0 )
141     {
142         psp.set(GI_TOP_LABEL,sprintf(buffer,"Solution at time t=%e",t)); // set title
143         PlotIt::contour(ps, u,psp );
144     }
145
146     getIndex(mg.dimension(),I1,I2,I3);
147     u+=dt*( (-a)*u.x() +(-b)*u.y() +nu*(u.xx() +u.yy() ) +FORCE(x(I1,I2,I3,0),x(I1,I2,I3,1),t) );
148     t+=dt;
149     // apply Boundary conditions
150     // apply Boundary conditions
151     for( int axis=0; axis<mg.numberOfDimensions(); axis++ )
152         for( int side=Start; side<=End; side++ )
153         { // only assign BC's on sides with a positive boundary condition:
154             if( mg.boundaryCondition()(side,axis) > 0 )
155                 { // fill in boundary values
156                     getBoundaryIndex(mg.gridIndexRange(),side,axis,Ib1,Ib2,Ib3);
157                     u(Ib1,Ib2,Ib3)=UTRUE(x(Ib1,Ib2,Ib3,0),x(Ib1,Ib2,Ib3,1),t);
158                 }
159         }
160     u.periodicUpdate(); // swap periodic edges
161
162     getIndex(mg.gridIndexRange(),I1,I2,I3);
163     real error = max(abs( u(I1,I2,I3)-UTRUE(x(I1,I2,I3,0),x(I1,I2,I3,1),t) ));
164     cout << "t=" << t << ", error =" << error << endl;
165 }
166
167 Overture::finish();
168 return 0;
169 }
170

```

2.8 Example: Steady state, linearized, incompressible Navier-Stokes Equations

This example shows how to solve a system of equations that looks something like the steady state incompressible Navier-Stokes equations. The system we solve is

$$\begin{aligned}\nu\Delta u - (u_0(x, y)u_x + v_0(x, y)u_y + p_x) &= f_0 \\ \nu\Delta v - (u_0(x, y)v_x + v_0(x, y)v_y + p_y) &= f_1 \\ \Delta p - \delta(u_x + v_y) &= f_2\end{aligned}$$

with boundary conditions

wall:	$u = \text{given}$	$v = \text{given}$	$p_n = \text{given}$
inflow:	$u = \text{given}$	$v = \text{given}$	$p = \text{given}$
outflow:	$u_n = \text{given}$	$v_n = \text{given}$	$p = \text{given}$

The damping factor δ helps to keep $u_x + v_y$ small, see the papers [3],[16] for more details on discretizing the incompressible Navier-Stokes equations.

The system of equations is built using coefficient matrices, see the operator documentation, available from the Overture home page, for other examples. The program solves the equations on a square with an inflow boundary on the left, an outflow boundary on the right and no-slip walls on the top and bottom. The forcing functions f_0 , f_1 and f_2 are chosen so that the exact solution is known. This known solution is a quadratic polynomial for which the method should give the exact answer. Indeed if you run this example you should see that the computed errors are “zero” (round off level).

(file Overture/primer/lins.C)

3 Getting Started with CompositeGrid's

A CompositeGrid is a class that holds an overlapping grid. An overlapping grid can be created with the interactive grid generation program ogen, and saved in a data-base (HDF) file. Application programs such as the examples that follow can easily read the data-base file to create an overlapping grid.

3.1 Example 1: CompositeGrid's and MappedGrid's

Here is an example of how to create a CompositeGrid from a data base file created by the interactive grid generation program ogen.

(file Overture/primer/example1.C)

```

1 #include "Overture.h"
2
3 int
4 main(int argc, char *argv[])
5 {
6     Overture::start(argc,argv); // initialize Overture
7
8     printf(" ----- \n");
9     printf(" Read an overlapping grid from a data base file \n");
10    printf(" Loop over the component grids and display the boundary \n");
11    printf("   conditions and the grid points (vertex array) \n");
12    printf(" ----- \n");
13
14    aString nameOfOGFile;
15    cout << "Enter the name of the overlapping grid data base file " << endl;
16    cin >> nameOfOGFile;
17
18    // create and read in a CompositeGrid
19    CompositeGrid cg;
20    getFromADataBase(cg,nameOfOGFile);
21    cg.update();
22
23    for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ ) // loop over component grids
24    {
25        cg[grid].boundaryCondition().display("Here are the boundary conditions");
26        cg[grid].vertex().display("Here are the vertex coordinates");
27
28        // A Composite grid is a list of MappedGrid's. To save typing we can
29        // make a reference (alias):
30        MappedGrid & mg = cg[grid]; // make a reference to the MappedGrid
31        mg.boundaryCondition().display("Here is boundaryCondition again"); // same result as above
32    }
33
34    Overture::finish();
35    return 0;
36 }
37

```

We first read in a CompositeGrid from the data base file that was created with ogen. We then loop over the component grids and print out some variables. A component grid is actually a “MappedGrid”, as shown in the example. A MappedGrid is so named since it contains a mapping function. See sections 6 and 7 for a brief description of the variables that are contained in a MappedGrid and a CompositeGrid.

When I run this example the program will prompt for the name of the overlapping grid data base file, and I will enter the name of the file that I created with ogen, for example /home/henshaw/Overture/cgsh/square5.hdf.

The file Overture/primer/gridQuery.C is a program that can be used to read in and display various information about a CompositeGrid. .

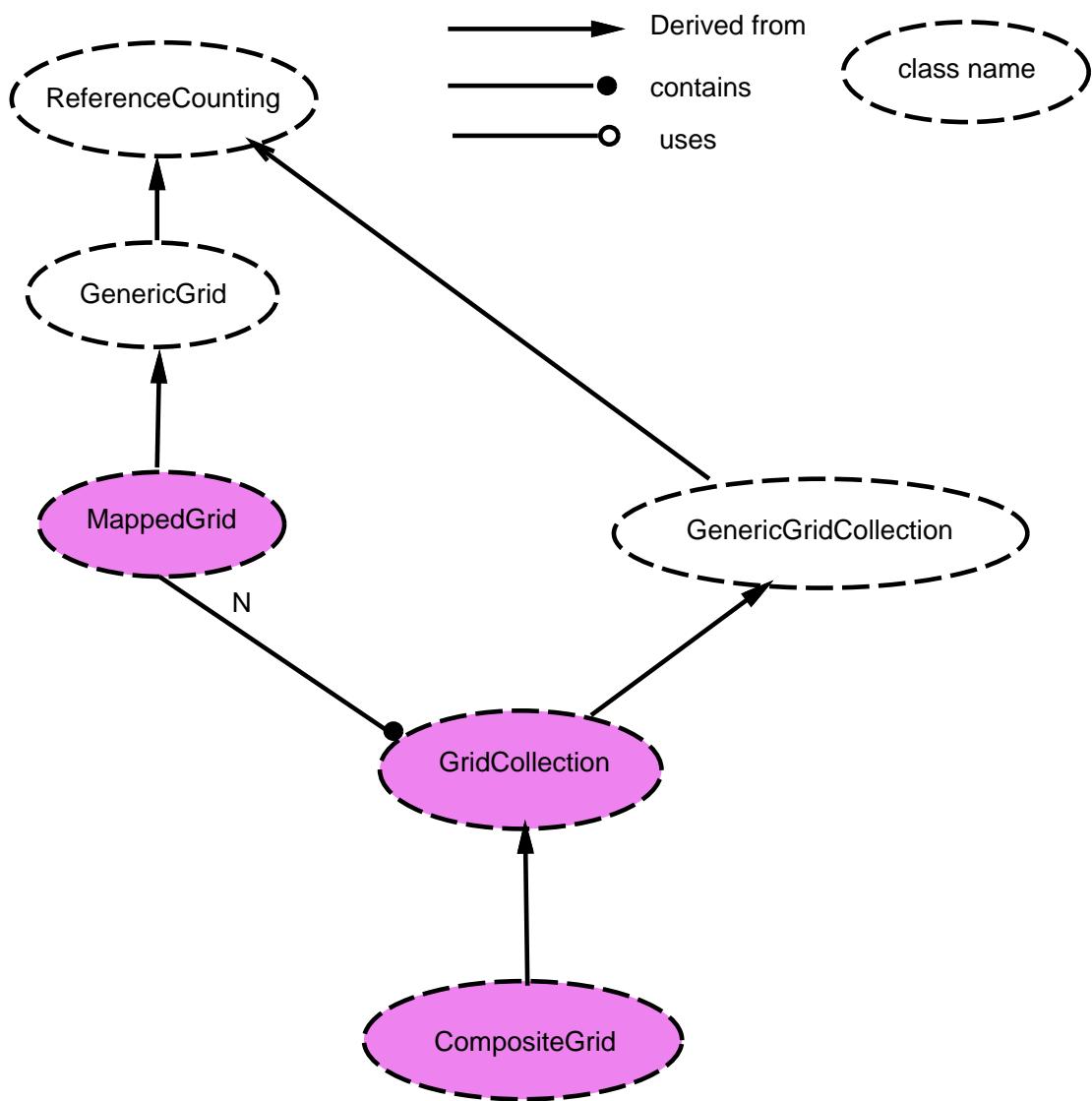
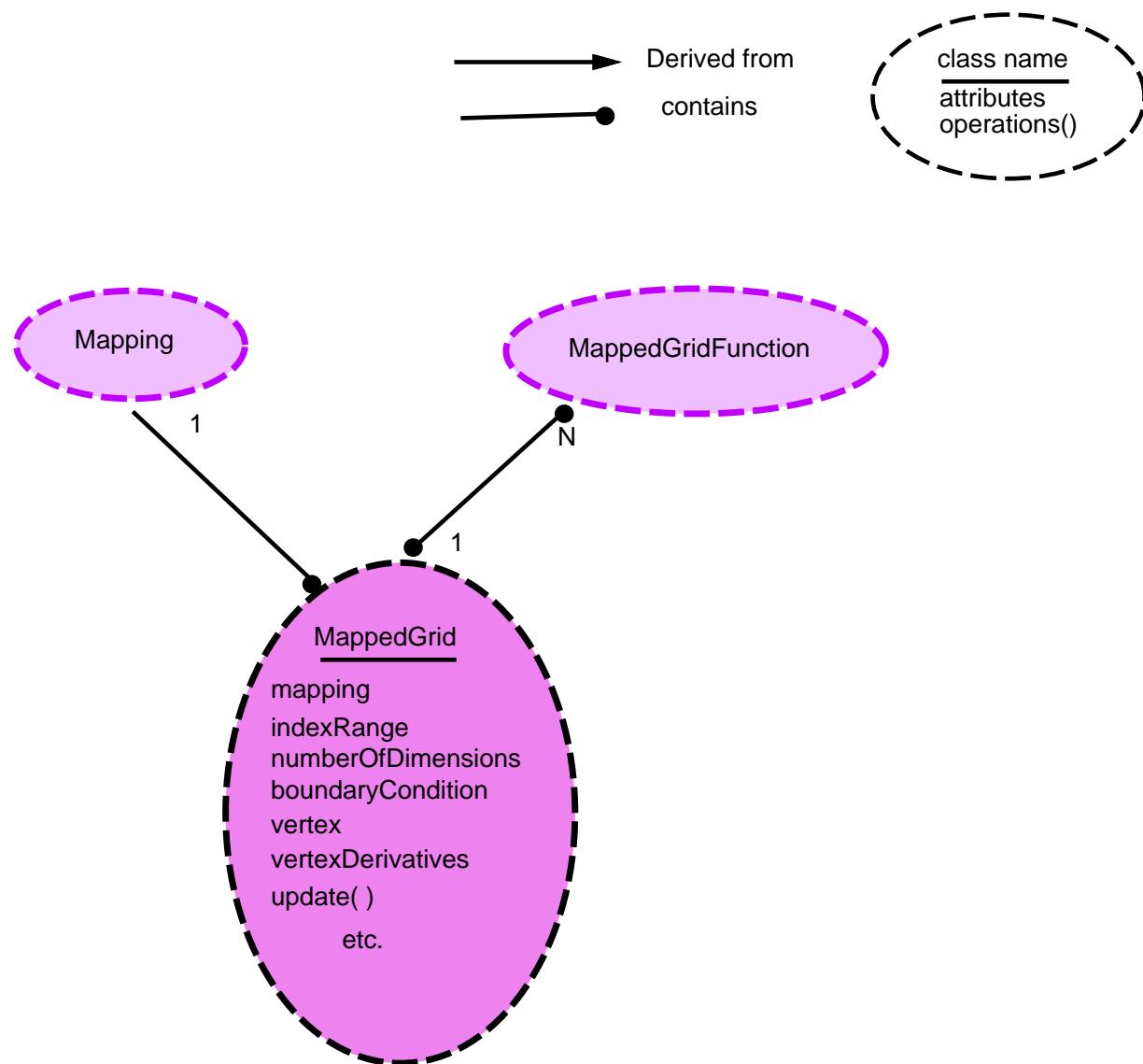


Figure 4: Class diagram for grid classes

Figure 5: Class diagram for a `MappedGrid`

3.2 Example 2: grid functions

In the next example we introduce the notion of a grid function. A “realCompositeGridFunction” is a discrete function that lives on the grid points (or cell centres or faces) of a CompositeGrid [5]. A realCompositeGridFunction contains a list of “realMappedGridFunctions”, one realMappedGridFunction for each component grid.

(file Overture/primer/example2.C)

```

1  #include "Overture.h"
2
3  int
4  main(int argc, char *argv[])
5  {
6      Overture::start(argc,argv); // initialize Overture
7
8      printf(" ----- \n");
9      printf(" Read an overlapping grid from a data base file \n");
10     printf(" Make a grid function, assign it values and then display it \n");
11     printf(" ----- \n");
12
13     aString nameOfOGFile;
14     cout << "Enter the name of the overlapping grid data base file " << endl;
15     cin >> nameOfOGFile;
16
17     // create and read in a CompositeGrid
18     CompositeGrid cg;
19     getFromADataBase(cg,nameOfOGFile);
20     cg.update();
21
22     realCompositeGridFunction u(cg); // create a composite grid function
23     u=0.; // initialize to zero
24     Index I1,I2,I3; // A++ Index object
25
26     for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ ) // loop over component grids
27     {
28         getIndex(cg[grid].indexRange(),I1,I2,I3); // assign I1,I2,I3 from indexRange
29         u[grid](I1,I2,I3)=sin(cg[grid].vertex()(I1,I2,I3,axis1)) // assign all interior points on this
30             *cos(cg[grid].vertex()(I1,I2,I3,axis2)); // component grid
31     }
32     u.display("here is u=sin(x)*cos(y)");
33
34     Overture::finish();
35     return 0;
36 }
```

A “realCompositeGridFunction will either be a grid function of “floats” or a grid function of “doubles” depending on a compiler flag. In the above example we use the function getIndex to define the Index objects I1, I2, I3 corresponding to the indexRange – i.e. the interior points of the grid. The interior points on each component grid of the grid function u are given values equal to $\sin(x)\cos(y)$. The object u[grid] is a “realMappedGridFunction”. This object is derived from an A++ array and thus inherits all the functionality of an A++ array.

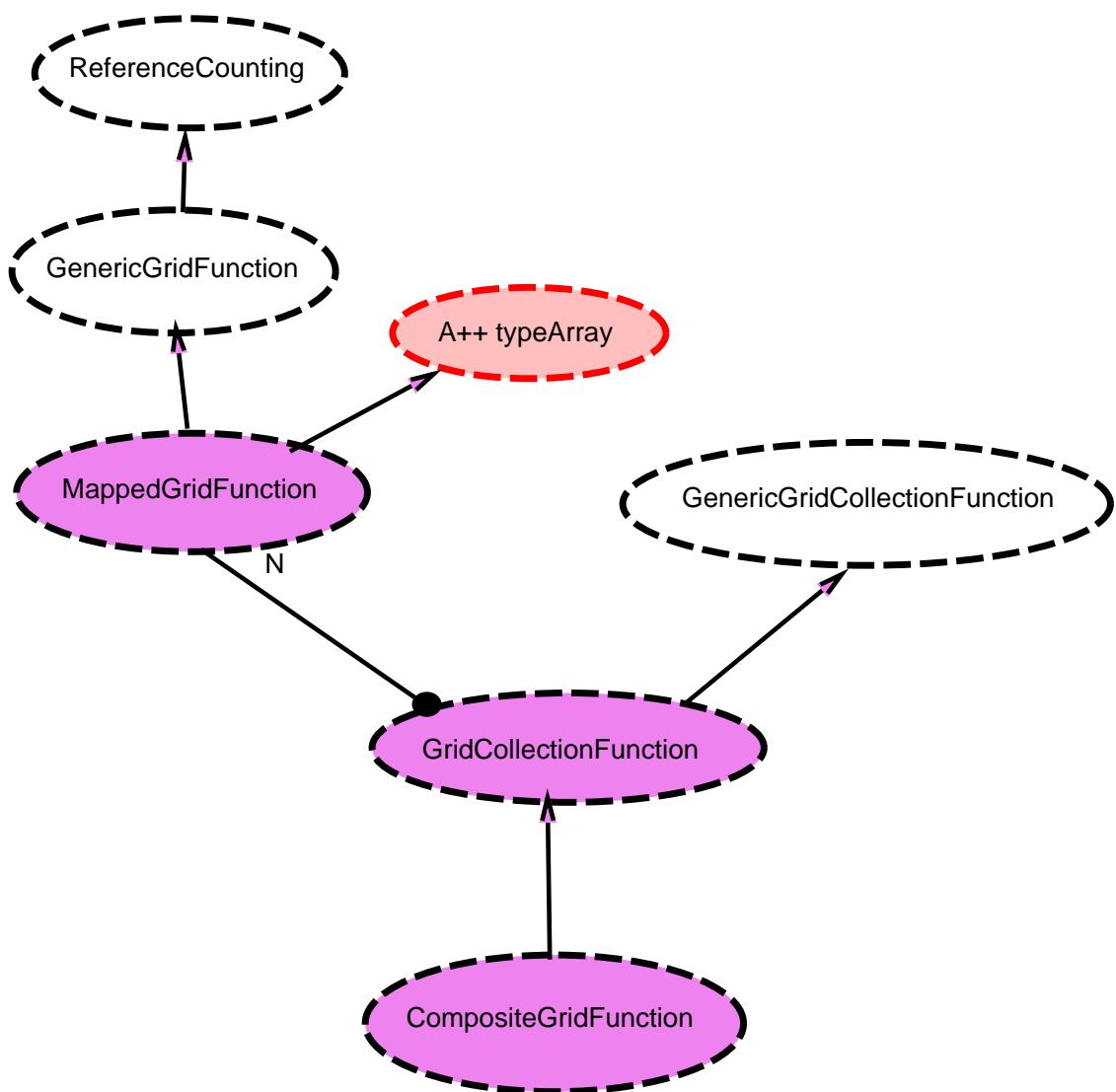
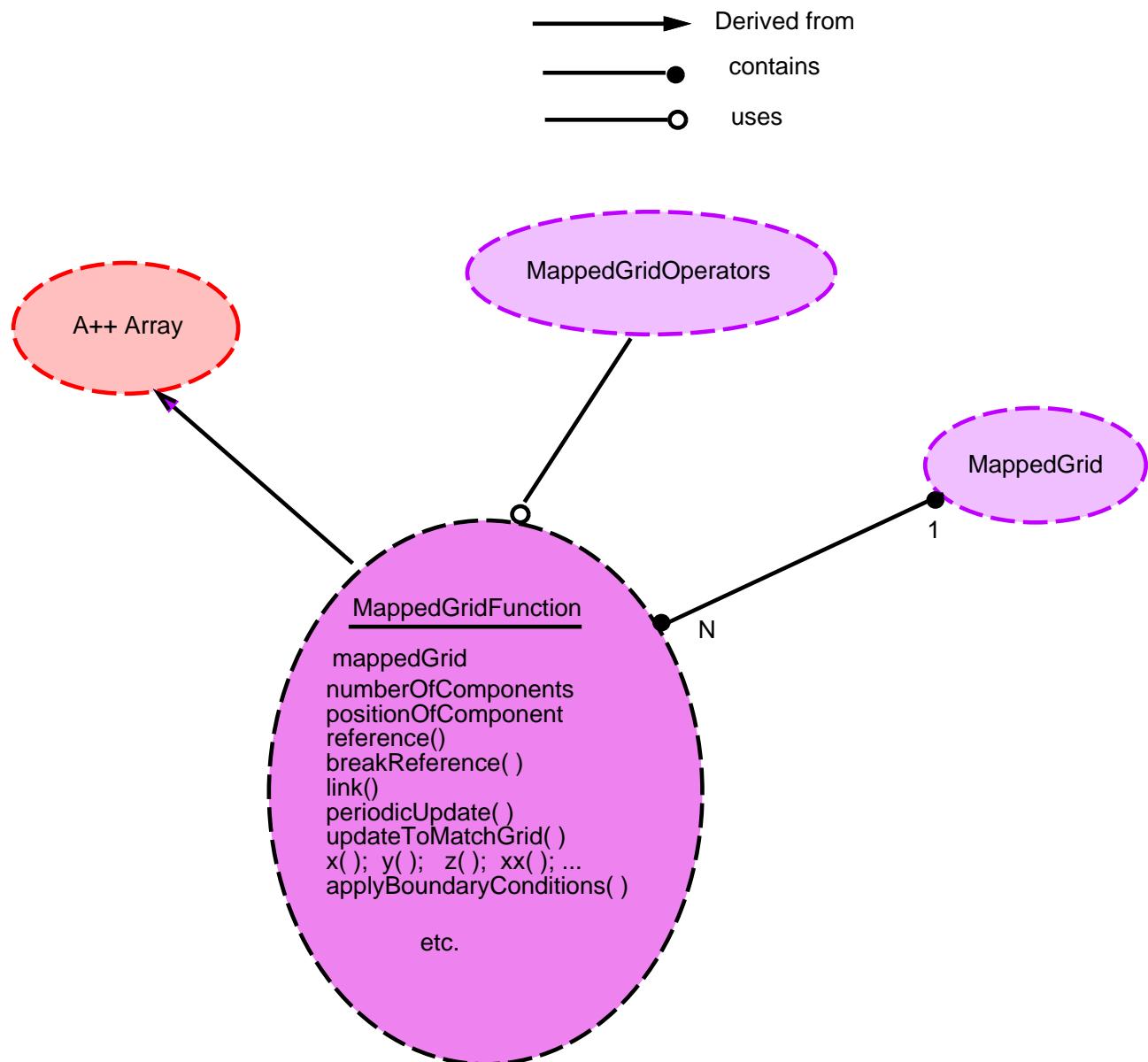


Figure 6: Class diagram for grid function classes

Figure 7: Class diagram for a `MappedGridFunction`

3.3 Example 3: interpolation

In the next example we show how to interpolate a grid function, i.e. how to obtain the values at the interpolation points given the values at all other points. In order to interpolate you must first create an “Interpolant” object. This object knows how to interpolate grid functions on a given CompositeGrid.

(file Overture/primer/example3.C)

```

1  #include "Overture.h"
2
3  int
4  main(int argc, char *argv[])
5  {
6      Overture::start(argc,argv); // initialize Overture
7
8      printf(" ----- \n");
9      printf(" Demonstrate how to interpolate a compositeGridFunction \n");
10     printf(" ----- \n");
11
12     aString nameOfOGFile;
13     cout << "Enter the name of the overlapping grid data base file " << endl;
14     cin >> nameOfOGFile;
15
16     // create and read in a CompositeGrid
17     CompositeGrid cg;
18     getFromADataBase(cg,nameOfOGFile);
19     cg.update();
20
21     realCompositeGridFunction u(cg);           // create a composite grid function
22     u=0.;                                     // initialize to zero
23     Index I1,I2,I3;                          // A++ Index object
24
25     for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ ) // loop over component grids
26     {
27         getIndex(cg[grid].indexRange(),I1,I2,I3);           // assign I1,I2,I3
28         where( cg[grid].mask()(I1,I2,I3) > 0 )             // only assign points with mask>0
29         u[grid](I1,I2,I3)=sin(cg[grid].vertex()(I1,I2,I3,axis1)) // do not assign interpolation points
30             *cos(cg[grid].vertex()(I1,I2,I3,axis2));
31     }
32     u.display("here is u=sin(x)*cos(y) before interpolation");
33
34     Interpolant interpolant(cg);        // Make an interpolant
35     interpolant.interpolate(u);         // interpolate
36     u.display("here is u after interpolation");
37
38     u.interpolate();      // another way to interpolate, same result as above
39     u.display("here is u after interpolate, version 2");
40
41     Overture::finish();
42     return 0;
43 }
```

In this example we use the mask array to selectively assign the grid points. The mask array is positive for discretization points, negative for interpolation points and zero for unused points. After interpolation the values at points with $\text{mask} < 0$ will have been assigned. As shown in the example there are two ways to interpolate. The second way, `u.interpolate()` may seem a bit mysterious since why should the grid function know about the Interpolant? The answer is that when the Interpolant is made it tells the CompositeGrid that it exists. The grid function checks with the CompositeGrid that it is associated with to see if an Interpolant has been made and if so it uses it.

3.4 Example 4: show files

Grid functions can be saved in a “show file” and later displayed with plotStuff (in the Overture/bin directory). In this example we show how to make a show file. More work has to be done on show files so some of the syntax may change in the future.
(file Overture/primer/example4.C)

```

1  #include "Overture.h"
2  #include "Ogshow.h"
3
4  int
5  main(int argc, char *argv[])
6  {
7      Overture::start(argc,argv); // initialize Overture
8
9      printf(" ----- \n");
10     printf(" Demonstrate how to use the Ogshow class to save results in a file \n");
11     printf(" to be later plotted with plotStuff \n");
12     printf(" ----- \n");
13
14     aString nameOfOGFile, nameOfShowFile;
15     cout << "example4>> Enter the name of the (old) overlapping grid file:" << endl;
16     cin >> nameOfOGFile;
17     cout << "example4>> Enter the name of the (new) show file (blank for none):" << endl;
18     cin >> nameOfShowFile;
19
20     // create and read in a CompositeGrid
21     CompositeGrid cg;
22     getFromADataBase(cg,nameOfOGFile);
23     cg.update();
24
25     Ogshow show( nameOfShowFile ); // create a show file
26
27     show.saveGeneralComment("Solution to the Navier-Stokes"); // save a general comment in the show file
28     show.saveGeneralComment(" file written on April 1"); // save another general comment
29
30     Range all; // a null Range is used to dimension the grid function
31     const int numberOfComponents=3;
32     realCompositeGridFunction q(cg,all,all,all,numberOfComponents); // create a grid function with 3 components
33     q=0.;
34
35     realCompositeGridFunction u,v,machNumber; // create grid functions for components
36     u.link(q,Range(0,0)); // link u to the first component of q
37     v.link(q,Range(1,1)); // link v to the second component of q
38     machNumber.link(q,Range(2,2)); // ...
39     q.setName("q"); // assign name to grid function and components
40     q.setName("u",0); // name of first component
41     q.setName("v",1); // name of second component
42     q.setName("Mach Number",2); // name of third component
43
44     char buffer[80]; // buffer for sPrintf
45     Index I1,I2,I3;
46     int numberofTimeSteps=5;
47     for( int i=1; i<=numberofTimeSteps; i++ ) // Now save the grid functions at different time steps
48     {
49         show.startFrame(); // start a new frame
50         real t=i*.1;
51         show.saveComment(0,sprintf(buffer,"Here is solution %i",i)); // comment 0 (shown on plot)
52         show.saveComment(1,sprintf(buffer," t=%e ",t)); // comment 1 (shown on plot)
53         for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ ) // loop over component grids
54         {
55             getIndex(cg[grid].indexRange(),I1,I2,I3);
56             u[grid](I1,I2,I3)=sin(twoPi*(cg[grid].vertex()(I1,I2,I3,axis1)-t)) // assign u on each grid
57                         *cos(twoPi*(cg[grid].vertex()(I1,I2,I3,axis2)+t));
58         }
59         v=u*2.;
60         machNumber=u*u+v*v;
61         show.saveSolution( q ); // save the current grid function
62     }
63
64     Overture::finish();
65     return 0;
66

```

```
67 }
```

This example demonstrates a few other features of grid functions such as declaring a grid function with more than one component and linking one grid function to another.

Use plotStuff to display the results from this example, “plotStuff fileName.show” where “fileName.show” was the name chosen for the showfile when the program was run.

3.5 Example 5: Differentiating grid functions

The MappedGridOperators and CompositeGridOperators classes can be used to compute spatial derivatives of grid functions and to apply boundary conditions. In this example we show how to differentiate grid functions to second or fourth-order accuracy.

(file Overture/primer/example5.C)

```

1  #include "Overture.h"
2  #include "CompositeGridOperators.h"
3
4  int
5  main(int argc, char *argv[])
6  {
7      Overture::start(argc,argv); // initialize Overture
8
9      printf(" ----- \n");
10     printf(" Demonstrate the operators for taking derivatives of compositeGridFunction's \n");
11     printf(" ----- \n");
12
13     aString nameOfOGFile;
14     cout << "example5>> Enter the name of the (old) overlapping grid file:" << endl;
15     cin >> nameOfOGFile;
16
17     // create and read in a CompositeGrid
18     CompositeGrid cg;
19     getFromADataBase(cg,nameOfOGFile);
20     cg.update();
21
22     CompositeGridOperators operators(cg); // operators for a CompositeGridFunction
23     Range all;
24     realCompositeGridFunction u(cg),ux(cg),w(cg,all,all,all,2); // create some composite grid functions
25
26     u.setOperators(operators); // tell grid function which operators to use
27     w.setOperators(operators);
28
29     u=1.;
30     ux=u.x(); // compute the x derivative of u
31     ux.display("Here is the x derivative of u=1 (computed at interior and boundary points)");
32     w=2.;
33     w.y().display("Here is the y derivative of w");
34     Range c0(0,0),c1(1,1);
35     w.y(c0).display("Here is the y derivative of component 0 of w");
36     w.y(c1).display("Here is the y derivative of component 1 of w");
37
38     real error;
39     Index I1,I2,I3;
40     for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ ) // loop over component grids
41     {
42         MappedGrid & mg = cg[grid];
43         getIndex(mg.dimension(),I1,I2,I3); // assign I1,I2,I3 for dimension
44         u[grid](I1,I2,I3)=sin(mg.vertex()(I1,I2,I3, axis1))*cos(mg.vertex()(I1,I2,I3, axis2));
45         getIndex(mg.indexRange(),I1,I2,I3); // assign I1,I2,I3 for indexRange
46         operators.setOrderOfAccuracy(2); // set order of accuracy to 4
47
48         ux[grid](I1,I2,I3)=u[grid].x()(I1,I2,I3); // here is the x derivative of u[grid]
49
50         error = max(fabs( ux[grid](I1,I2,I3)- cos(mg.vertex()(I1,I2,I3, axis1))*cos(mg.vertex()(I1,I2,I3, axis2) )));
51         cout << "Maximum error (2nd order) = " << error << endl;
52
53         error = max(fabs( operators[grid].x(u[grid])(I1,I2,I3) // another way to compute derivatives
54                         - cos(mg.vertex()(I1,I2,I3, axis1))*cos(mg.vertex()(I1,I2,I3, axis2) )));
55         cout << "Maximum error (2nd order) = " << error << endl;
56
57         operators.setOrderOfAccuracy(4); // set order of accuracy to 4
58         getIndex(mg.indexRange(),I1,I2,I3,-1); // decrease ranges by 1 for 4th order
59         error = max(fabs(u[grid].x()(I1,I2,I3)-cos(mg.vertex()(I1,I2,I3, axis1))*cos(mg.vertex()(I1,I2,I3, axis2))));
60         cout << "Maximum error (4th order) = " << error << endl;
61     }
62     Overture::finish();
63     return 0;
64 }
```

In this example we create a `CompositeGridOperators` object and associate it with a `CompositeGrid`. We compute the x-derivative of a `realCompositeGridFunction` and of `realMappedGridFunction`'s. The member function "x" in the grid function returns the x derivative of the grid function as a new grid function. It uses the derivative defined in the `CompositeGridOperators` object which in turn uses a `MappedGridOperators` object to compute the derivatives of a `MappedGridFunction`. The default `MappedGridOperators` object used by a `CompositeGridOperators` can be changed. Note that by default the derivative of a `realCompositeGridFunction` is only computed at interior and boundary points (`indexRange`). Thus to access (make a view) of the derivative values of the grid function `u.x()` at the Index's `(I1, I2, I3)` it is necessary to say `u.x()(I1, I2, I3)`. On the other hand the statement `u.x(I1, I2, I3)` will evaluate the derivatives on the points defined by `(I1, I2, I3)`, but will return a grid function that is dimensioned for the entire grid. Thus in general one could say `u.x(I1, I2, I3)(J1, J2, J3)` to evaluate the derivatives at points `(I1, I2, I3)` but to use (take a view) of the grid function at the Index's `(J1, J2, J3)`.

The `MappedGridOperators` and `CompositeGridOperators` classes are described in more detail in the grid function documentation.

3.6 Example 6: Solving a simple PDE using Differential and Boundary operators

In this example we solve the convection-diffusion equation

$$\frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} + b \frac{\partial u}{\partial y} = \nu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

with a simple time stepping method (forward Euler) The solutions at different time steps are saved in a show file.
(file Overture/primer/example6.C)

```

1  #include "Overture.h"
2  #include "Ogshow.h"
3  #include "CompositeGridOperators.h"
4
5  int
6  main(int argc, char *argv[])
7  {
8      Overture::start(argc,argv); // initialize Overture
9
10     printf(" ----- \n");
11     printf("Solve: u.t + a*u.x + b*u.y = viscosity*( u.xx + u.yy ) on an Overlapping grid \n");
12     printf("Save results in a show file, use plotStuff to view this file \n");
13     printf(" ----- \n");
14
15     aString nameOfOGFile, nameOfShowFile;
16     cout << "example6>> Enter the name of the (old) overlapping grid file:" << endl;
17     cin >> nameOfOGFile;
18     cout << "example6>> Enter the name of the (new) show file (blank for none):" << endl;
19     cin >> nameOfShowFile;
20
21     // create and read in a CompositeGrid
22     CompositeGrid cg;
23     getFromADataBase(cg,nameOfOGFile);
24     cg.update();
25
26     Interpolant interpolant(cg); // Make an interpolant
27
28     Ogshow show( nameOfShowFile );
29     show.saveGeneralComment("Convection Diffusion Equation");
30     // show.setFlushFrequency(10); // save a general comment in the show file
31                                         // flush file every 10 frames
32
33     CompositeGridOperators operators(cg); // operators for a CompositeGrid
34     // operators.setOrderOfAccuracy(4); // for fourth order
35
36     Range all;
37     realCompositeGridFunction u(cg,all,all,all,1); // create a grid function
38     u.setName("u"); // name the grid function
39
40     u=1.; // initial condition
41     real t=0, dt=.001; // initialize time and time step
42     real a=1., b=1., viscosity=.1; // initialize parameters
43
44     char buffer[80]; // buffer for sprintf
45     int numberoftimesteps=200;
46     for( int i=0; i<numberoftimesteps; i++ ) // take some time steps
47     {
48         if( i % 40 == 0 ) // save solution every 10 steps
49         {
50             show.startFrame(); // start a new frame
51             show.saveComment(0,sprintf(buffer,"Here is solution %i",i)); // comment 0 (shown on plot)
52             show.saveComment(1,sprintf(buffer," t=%e ",t)); // comment 1 (shown on plot)
53             show.saveSolution( u ); // save the current grid function
54         }
55         u+=dt*(-a*u.x() - b*u.y() + viscosity*(u.xx() + u.yy())); // take a time step with Euler's method
56         t+=dt;
57         u.interpolate(); // interpolate
58         // apply a dirichlet BC on all boundaries:
59         u.applyBoundaryCondition(0,BCTypes::dirichlet,BCTypes::allBoundaries,0.);
60         // u.applyBoundaryCondition(0,BCTypes::extrapolate,BCTypes::allBoundaries,0.); // for 4th order
61         u.finishBoundaryConditions();
62     }

```

```
63
64     Overture::finish();
65     return 0;
66
67 }
```

To run this example:

- First create an overlapping grid. For example, from the Overture/sampleGrids directory type `../bin/ogen noplot cic.cmd` to create the grid `cic.hdf` using the command file `cic.cmd`.
- Run `example6`. Enter `cic.hdf` as the overlapping grid to use and `primer.show` as the name of the output “show” file.
- Look at the results by typing `../bin/plotStuff primer.show`.

Note that a fixed time step is used in this example and that the time step may not be small enough to keep the method stable if you use a different grid from `cic.hdf`.

Currently, computing derivatives in this way will not be so efficient. An efficient way to compute derivatives is described in section (3.10) and in the grid function documentation.

In this example we chose the boundary conditions to be dirichlet on all sides of all grids. By default the values at dirichet boundaries are set to zero. Boundary conditions can be defined in a much more general manner as described in the grid function documentation.

If you want to solve the problem with fourth-order accuracy you can un-comment the two lines indicated in `example6.C`. You will need to use an overlapping grid created for fourth order (such as `cic.4.cmd`) and you will need to decrease the time step `dt` by a factor of 4 or so(?)

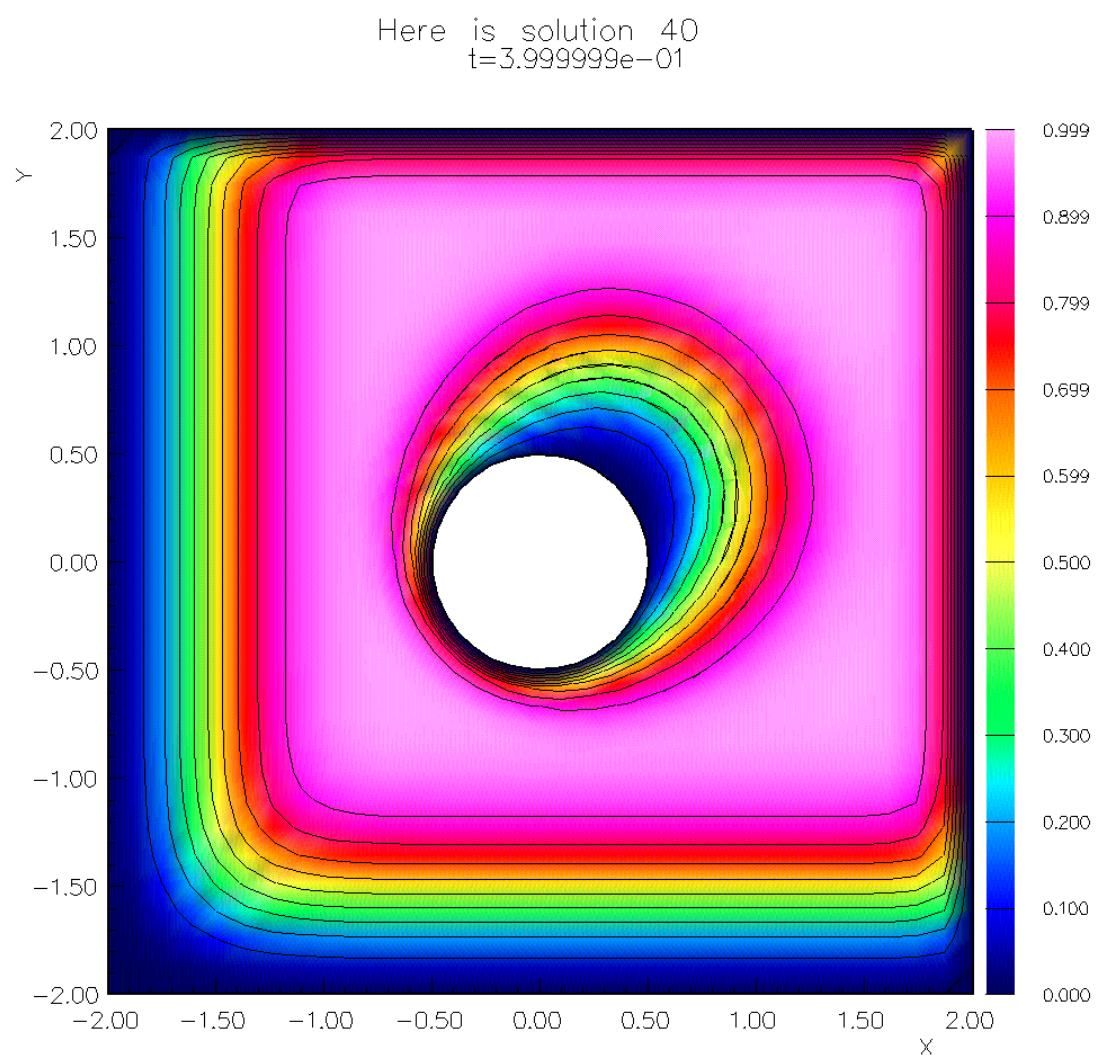


Figure 8: Results from example6. Solving a convection diffusion equation.

3.7 Example 7: Solving Poisson's equation with Oges

In this example we solve Poisson's equation in 2 or 3D,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = f \quad \text{for } \mathbf{x} \in \Omega$$

with Dirichlet boundary conditions

$$u = 0 \quad \text{for } \mathbf{x} \in \partial\Omega$$

(file Overture/primer/example7.C)

```

1  #include "Overture.h"
2  #include "CompositeGridOperators.h"
3  #include "Oges.h"
4
5  int
6  main(int argc, char *argv[])
7  {
8      Overture::start(argc,argv); // initialize Overture
9
10     printf(" ----- \n");
11     printf("Use the operators to create the matrix for the discrete Laplacian operator with\n");
12     printf(" boundary conditions.\n");
13     printf("Use the Oges class to solve the system of equations.\n");
14     printf(" ----- \n");
15
16     aString nameOfOGFile;
17     cout << "example7>> Enter the name of the (old) overlapping grid file:" << endl;
18     cin >> nameOfOGFile;
19
20     // create and read in a CompositeGrid
21     CompositeGrid cg;
22     getFromADataBase(cg,nameOfOGFile);
23     cg.update();
24
25     // make a grid function to hold the coefficients
26     Range all;
27     int stencilSize=int( pow(3,cg.numberofDimensions())+1.5 ); // add 1 for interpolation equations
28     realCompositeGridFunction coeff(cg,stencilSize,all,all,all);
29     coeff.setIsACoefficientMatrix(TRUE,stencilSize);
30
31     // create grid functions:
32     realCompositeGridFunction u(cg),f(cg);
33
34     CompositeGridOperators op(cg); // create some differential operators
35     op.setStencilSize(stencilSize);
36     coeff.setOperators(op);
37
38     coeff=op.laplacianCoefficients(); // get the coefficients for the Laplace operator
39     // make some shorter names for readability
40     BCTypes::BCNames dirichlet = BCTypes::dirichlet,
41             extrapolate = BCTypes::extrapolate,
42             allBoundaries = BCTypes::allBoundaries;
43
44     // fill in the coefficients for the boundary conditions
45     coeff.applyBoundaryConditionCoefficients(0,0,dirichlet, allBoundaries);
46     coeff.applyBoundaryConditionCoefficients(0,0,extrapolate,allBoundaries); // extrap ghost line
47     coeff.finishBoundaryConditions();
48
49     Oges solver( cg ); // create a solver
50     // solver.set(OgesParameters::THEkeepSparseMatrix,true); // turn this on if we want to save the matrix
51
52     solver.setCoefficientArray( coeff ); // supply coefficients
53
54
55     // assign the rhs: Laplacian(u)=1, u=0 on the boundary
56     Index I1,I2,I3;
57     Index Ib1,Ib2,Ib3;
58     for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ )
59     {
60         MappedGrid & mg = cg[grid];

```

```

61     getIndex(mg.indexRange(),I1,I2,I3);
62
63     f[grid](I1,I2,I3)=1.;
64     for( int side=Start; side<=End; side++ )
65     for( int axis=axis1; axis<cg.numberOfDimensions(); axis++ )
66     {
67         if( mg.boundaryCondition()(side,axis) > 0 )
68         {
69             getBoundaryIndex(mg.gridIndexRange(),side,axis,Ib1,Ib2,Ib3);
70             f[grid](Ib1,Ib2,Ib3)=0.;
71         }
72     }
73 }
74
75 solver.solve( u,f ); // solve the equations
76
77 // solver.writeMatrixToFile("matrix.dat"); // save the sparse matrix to a file (uncomment the line above too)
78
79
80 u.display("Here is the solution to Laplacian(u)=1, u=0 on the boundary");
81
82 Overture::finish();
83 return(0);
84
85 }
```

We use the Oges (Overlapping grid equation solver) class to use a sparse matrix solver to solve the problem. We use the differential operators in the CompositeGridOperators class to define coefficients of the Laplacian operator and the coefficients for the boundary condition. By default the Oges solver will use the Yale sparse matrix solver. The first time the problem is solved the matrix will be factored. Subsequent calls to solve with different right-hand-sides will only involve a back-substitution. See the Oges documentation for further details on the many available options.

If instead of the Laplacian operator we wanted to define some other operator, say,

$$2\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + 3\frac{\partial u}{\partial x}$$

then we could have used the statement

```
coeff=2.*u.xxCoefficients()+u.yyCoefficients()+3.*u.xCoefficients();
```

3.8 Example 8: Interactive plotting with PlotStuff

In this example we show how to plot “stuff” interactively from a program using the `PlotStuff` class. These plotting routines are based on OpenGL and can run on many platforms. Currently on Sun’s I use Brian Paul’s Mesa library which is a public domain implementation of OpenGL that runs under X-windows. More information about plotting can be found in the document `/home/henshaw/Overture/ogshow/PlotStuff.tex`.

Here is an example code that uses the `PlotStuff` class to plot various objects from the Overture class (file `Overture/primer/example8.C`)

```

1 #include "Overture.h"
2 #include "PlotStuff.h"
3
4 int
5 main(int argc, char *argv[])
{
7     Overture::start(argc,argv); // initialize Overture
8
9     printf(" ----- \n");
10    printf("Demonstrate interactive plotting using the PlotStuff class \n");
11    printf("Make a menu and selectively plot the grid or contours or streamlines.\n");
12    printf(" ----- \n");
13
14    aString nameOfOGFile;
15    cout << "example>> Enter the name of the (old) overlapping grid file:" << endl;
16    cin >> nameOfOGFile;
17
18    // create and read in a CompositeGrid
19    CompositeGrid cg;
20    getFromADataBase(cg,nameOfOGFile);
21    cg.update();
22
23    Range all;
24    realCompositeGridFunction u(cg,all,all,all,2); // create a grid function with 2 components
25    u.setName("Velocity Stuff"); // give names to grid function ...
26    u.setName("u Stuff",0); // ...and components
27    u.setName("v Stuff",1);
28    Index I1,I2,I3;
29    for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ ) // loop over component grids
30    {
31        getIndex(cg[grid].dimension(),I1,I2,I3); // assign I1,I2,I3 from dimension
32        u[grid](I1,I2,I3,0)=sin(Pi*cg[grid].center()(I1,I2,I3,axis1)) // component 0 : sin(x)*cos(y)
33                    *cos(Pi*cg[grid].center()(I1,I2,I3,axis2));
34        u[grid](I1,I2,I3,1)=cos(Pi*cg[grid].center()(I1,I2,I3,axis1)) // component 1 : cos(x)*sin(y)
35                    *sin(Pi*cg[grid].center()(I1,I2,I3,axis2));
36    }
37
38    bool openGraphicsWindow=TRUE;
39    PlotStuff ps(openGraphicsWindow,"example8"); // create a PlotStuff object
40    PlotStuffParameters psp; // This object is used to change plotting parameters
41
42    aString answer;
43    aString menu[] = {
44        "!example8",
45        "contour", // Make some menu items
46        "stream lines",
47        "grid",
48        "read command file",
49        "save command file",
50        "erase",
51        "exit",
52        "" }; // empty string denotes the end of the menu
53    for(;;)
54    {
55        ps.getMenuItem(menu,answer); // put up a menu and wait for a response
56        if( answer=="contour" )
57        {
58            psp.set(GI_TOP_LABEL, "My Contour Plot"); // set title
59            PlotIt::contour(ps,u,psp); // contour/surface plots
60        }
61        else if( answer=="grid" )
62        {

```

```
63     PlotIt::plot(ps,cg);                                // plot the composite grid
64 }
65 else if( answer=="stream lines" )
66 {
67     PlotIt::streamLines(ps,u);                         // streamlines
68 }
69 else if( answer=="read command file" )
70 {
71     ps.readCommandFile();
72 }
73 else if( answer=="save command file" )
74 {
75     ps.saveCommandFile();
76 }
77 else if( answer=="erase" )
78 {
79     ps.erase();
80 }
81 else if( answer=="exit" )
82 {
83     break;
84 }
85 }
86 Overture::finish();
87 return 0;
88 }
89 }
90 }
```

When the example is run a window will pop up. To see the menus, put the cursor over the window and press the right mouse button. Choosing **contour**, for example, will cause the `contour` function to be called. Now choose **plot** to display the contour/surface plot. Other menu items allow one to change features of the plot. Buttons on the window allow one to shift rotate and zoom the plot. The left mouse button can be used to zoom using a rubber-band box.

3.9 Example 9: Saving and Reading a Restart file

This example show how to save information in a data base file. This could be a restart file for a PDE solver. It is also possible to create a hierarchical directory structure within the data base file. See the data base documentation for further details.

See also the documentation on ShowFileReader in the Ogshow documentation for how to read grids and grid functions from a show file. This would be another way to get initial conditions for a solver.

(file Overture/primer/example9.C)

```

1 #include "Overture.h"
2 #include "HDF_DataBase.h"
3
4 int
5 main(int argc, char *argv[])
6 {
7     Overture::start(argc,argv); // initialize Overture
8
9     printf(" ----- \n");
10    printf("This example shows how to save information in a data base file. \n");
11    printf("This could be a restart file for a PDE solver \n");
12    printf("See also the documentation on ShowFileReader in the Ogshow documentation \n");
13    printf("for how to read grids and grid functions from a show file. This would \n");
14    printf("be another way to get initial conditions for a solver. \n");
15    printf(" ----- \n");
16
17    aString nameOfOGFile;
18    cout << "example>> Enter the name of the (old) overlapping grid file:" << endl;
19    cin >> nameOfOGFile;
20
21    // create and read in a CompositeGrid
22    CompositeGrid cg;
23    getFromADataBase(cg,nameOfOGFile);
24    cg.update();
25
26    Range all;
27    realCompositeGridFunction u(cg,all,all,all,2); // create a grid function with 2 components
28
29    // u.setName("Velocity Stuff"); // give names to grid function ...
30    // u.setName("u Stuff",0); // ...and components
31    // u.setName("v Stuff",1);
32    Index I1,I2,I3;
33    for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ ) // loop over component grids
34    {
35        getIndex(cg[grid].dimension(),I1,I2,I3); // assign I1,I2,I3 from dimension
36        const realArray & x = cg[grid].center()(I1,I2,I3,axis1);
37        const realArray & y = cg[grid].center()(I1,I2,I3,axis2);
38
39        u[grid](I1,I2,I3,0)=sin(Pi*x)*cos(Pi*y); // component 0
40        u[grid](I1,I2,I3,1)=cos(Pi*x)*sin(Pi*y); // component 1
41    }
42    // Here are some other things that we want to save:
43    realArray drag(100);
44    drag=3.;
45    real time=55.6;
46    aString comment = "my restart file";
47
48    HDF_DataBase db; // make a data base
49
50    cout << "Mount file example9.hdf and save some data... \n";
51
52    db.mount("example9.hdf","I"); // open the data base, I=initialize
53
54    cg.put(db,"My Grid"); // save a grid
55    u.put(db,"My Solution"); // save a grid function
56    db.put(drag,"drag"); // save an array of data
57    db.put(time,"time"); // save a real number
58    db.put(comment,"comment"); // save a string
59
60    cout << "Close the file example9.hdf... \n";
61    db.unmount(); // close the data base
62
63

```

```
64 // Now mount the file and read back the data
65
66 HDF_DataBase db2;
67 cout << "Mount file example9.hdf and read back some data...\n";
68 db2.mount("example9.hdf", "R"); // mount, R=read-only
69
70 // define new objects to read the data into
71 CompositeGrid cg2;
72 realCompositeGridFunction u2;
73
74 realArray drag2;
75 real time2;
76 aString comment2;
77
78 // note that the data can be read back in any order
79 db2.get(drag2,"drag"); // save an array of data
80 db2.get(time2,"time"); // save a real number
81 db2.get(comment2,"comment"); // save a string
82
83 cg2.get(db2,"My Grid");
84
85 u2.updateToMatchGrid(cg2); // *** note: do an update before reading in the grid
86 u2.get(db2,"My Solution");
87 u2.display("Here is u2");
88
89 db2.unmount(); // close the file
90
91 // now check that we have read back the data properly
92
93 if( max(fabs(u2-u))==0. && max(abs(drag2-drag))==0. && fabs(time2-time)==0. && comment==comment2 )
94     cout << "Objects were successfully read back in\n" << endl;
95 else
96     cout << "ERROR: Objects were NOT successfully read back in\n" << endl;
97
98 Overture::finish();
99 return 0;
100 }
101 }
```

3.10 Example 10: Efficient computation of derivatives for PDE solvers

efficient computation, example

An efficient method for computing derivatives is shown in this example. See also the example for the wave equation (3.11) for a perhaps even more efficient approach.

One must first indicate how many derivatives will be evaluated, `setNumberOfDerivativesToEvaluate`, and which derivatives should be evaluated, `setDerivativeType`, and also supply A++ arrays to hold the results in (uvx , uvy , $uvxx$, $uvyy$). These arrays will automatically be made large enough to hold the results if they are not already large enough. The arrays are not, however, made smaller if they are too big. Thus once the arrays are large enough for the grid with the most grid points the arrays will not be redimensioned anymore. The call to `getDerivatives` will evaluate all the derivatives all at once (thus saving computations) and place the results in the user supplied arrays (thus saving memory allocation overhead).

(file Overture/primer/example10.C)

```

1  #include "Overture.h"
2  #include "PlotStuff.h"
3  #include "CompositeGridOperators.h"
4
5  int
6  main(int argc, char *argv[])
7  {
8      Overture::start(argc,argv); // initialize Overture
9
10     printf(" ----- \n");
11     printf("Solve a PDE using the efficient derivative evaluation \n");
12     printf("      u_t + u u_x + v u_y = nu( u.xx + u.yy ) \n");
13     printf("      v_t + u v_x + v v_y = nu( v.xx + v.yy ) \n");
14     printf(" ----- \n");
15
16     aString nameOfOGFile;
17     cout << "example>> Enter the name of the (old) overlapping grid file:" << endl;
18     cin >> nameOfOGFile;
19
20     // create and read in a CompositeGrid
21     CompositeGrid cg;
22     getFromADataBase(cg,nameOfOGFile);
23     cg.update();
24
25     Interpolant interpolant(cg); // Make an interpolant
26
27     PlotStuff ps(TRUE,"example10");
28     PlotStuffParameters psp;
29
30     CompositeGridOperators operators(cg); // operators for a CompositeGrid
31
32     Range all;
33     realCompositeGridFunction uv(cg,all,all,all,2); // create a grid function with 2 components
34     uv.setOperators(operators);
35     uv.setName("uv"); // name the grid function
36     uv.setName("u",0); // name the component
37     uv.setName("v",1); // name the component
38     realCompositeGridFunction u,v; // make links to the 2 components
39     u.link(uv,Range(0,0));
40     v.link(uv,Range(1,1));
41
42     // The arrays  $uvx$ ,  $uvy$ ,  $uvxx$  and  $uvyy$  are used to save the results in. These arrays are re-used for all
43     // the different component grids (thus saving space)
44     RealArray uvx,uvy,uvxx,uvyy;
45     // --- make a list of derivatives to evaluate on each component grid
46     int grid;
47     for( grid=0; grid<cg.numberOfComponentGrids(); grid++ )
48     {
49         operators[grid].setNumberOfDerivativesToEvaluate( 4 );
50         operators[grid].setDerivativeType( 0, MappedGridOperators::xDerivative, uvx );
51         operators[grid].setDerivativeType( 1, MappedGridOperators::yDerivative, uvy );
52         operators[grid].setDerivativeType( 2, MappedGridOperators::xxDerivative, uvxx );
53         operators[grid].setDerivativeType( 3, MappedGridOperators::yyDerivative, uvyy );
54     }
55
56     u=+1.; // initial condition u=+1

```

```

57     v=-1.;                                // initial condition  v=-1
58
59     real t=0, dt=.01;                      // initialize time and time step
60     real viscosity=.1;                     // initialize parameters
61
62     Index I1,I2,I3;                      // Index for components, N=0,1
63     Index N(0,2);                         // buffer for sprintf
64     char buffer[80];
65     int numberOfTimeSteps=250;
66     for( int i=0; i<=numberOfTimeSteps; i++ )           // take some time steps
67     {
68       if( i % 5 ==0 )
69     {
70       psp.set(GI_TOP_LABEL,sprintf(buffer,"Solution at t=%e",t));
71       ps.erase();
72       PlotIt::contour(ps,uv,psp);
73       psp.set(GI_PLOT_THE_OBJECT_AND_EXIT,TRUE);      // set this to run in "movie" mode (after first plot)
74       ps.redraw(TRUE);
75     }
76     for( grid=0; grid<cg.numberOfComponentGrids(); grid++ )
77     {
78       getIndex(cg[grid].gridIndexRange(),I1,I2,I3);    // define Index's for interior+boundary pts
79       uv[grid].getDerivatives(I1,I2,I3,N);             // evaluate all derivatives at once
80
81       RealArray & uu = u[grid];                        // make some aliases for readability
82       RealArray & vv = v[grid];                        // and efficiency
83       const RealArray & ux = uvx (I1,I2,I3,0);        // these values were computed by getDerivatives.
84       const RealArray & uy = uvy (I1,I2,I3,0);        // Note that these arrays will be redimensioned
85       const RealArray & uxx= uvxx(I1,I2,I3,0);        // by getDerivatives only if there is not enough
86       const RealArray & uyy= uvyy(I1,I2,I3,0);        // space. Thus after one time step the arrays
87       const RealArray & vx = uvx (I1,I2,I3,1);        // will be as large as the largest grid and then
88       const RealArray & vy = uvy (I1,I2,I3,1);        // will remain that size.
89       const RealArray & vxx= uvxx(I1,I2,I3,1);        //
90       const RealArray & vyy= uvyy(I1,I2,I3,1);        //
91
92       uu(I1,I2,I3)+=dt*(-uu(I1,I2,I3)*ux -vv(I1,I2,I3)*uy +viscosity*( uxx+uyy ));          // Euler time step
93       vv(I1,I2,I3)+=dt*(-uu(I1,I2,I3)*vx -vv(I1,I2,I3)*vy +viscosity*( vxx+vyy ));
94
95     }
96
97     uv.interpolate();                           // interpolate
98     // apply a dirichlet BC on all boundaries:
99     uv.applyBoundaryCondition(N,BCTypes::dirichlet,BCTypes::allBoundaries,0.);
100    uv.finishBoundaryConditions();
101    t+=dt;
102  }
103
104  Overture::finish();
105  return 0;
106 }
```

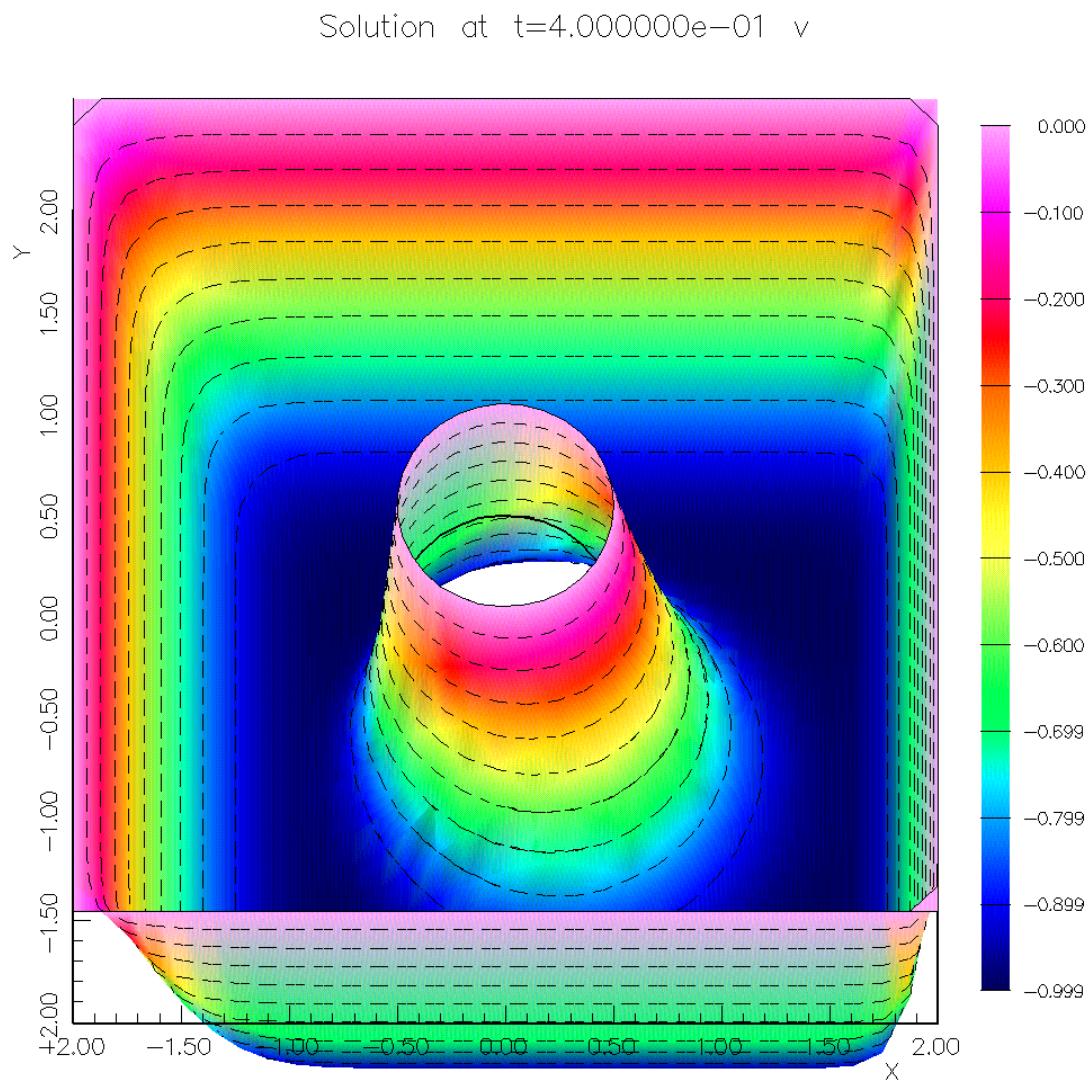


Figure 9: Results from example10, run with the grid cic.hdf

3.11 Example: 2D Wave equation (optimised for performance and memory usage)

This example shows how to solve the 2D wave equation,

$$u_{tt} - c^2 \Delta u = 0 .$$

This example has been optimised for performance and memory usage. Rectangular grids especially, are treated in an efficient manner.

We discretize this second order equation in time using a second-order centered difference,

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} - c^2 \Delta_h u_i^n = 0 ,$$

where the superscript n denotes the time level. We use either a 2nd order or 4th order discretization in space. We only interpolate assuming a second-order scheme so formally the method is not fourth-order.

The above scheme has no dissipation. To smooth out numerical oscillations an artificial dissipation term has been added of the form $h^4 \partial_t u_{xxxx}$,

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} - c^2 \Delta_h u_i^n + -C h^4 (D_+ D_-)^2 (u^n - u^{n-1}) = 0$$

To allow the use of a fourth-order difference approximation on an overlapping grid that is only built for a stencil that is 3×3 we use the `extrapolateInterpolationNeighbours` boundary condition to extrapolate values at the normally unused points next to interpolation points. With these values defined we can apply a fourth order difference at all interior points.

See file `Overture/primer/wave.C`.

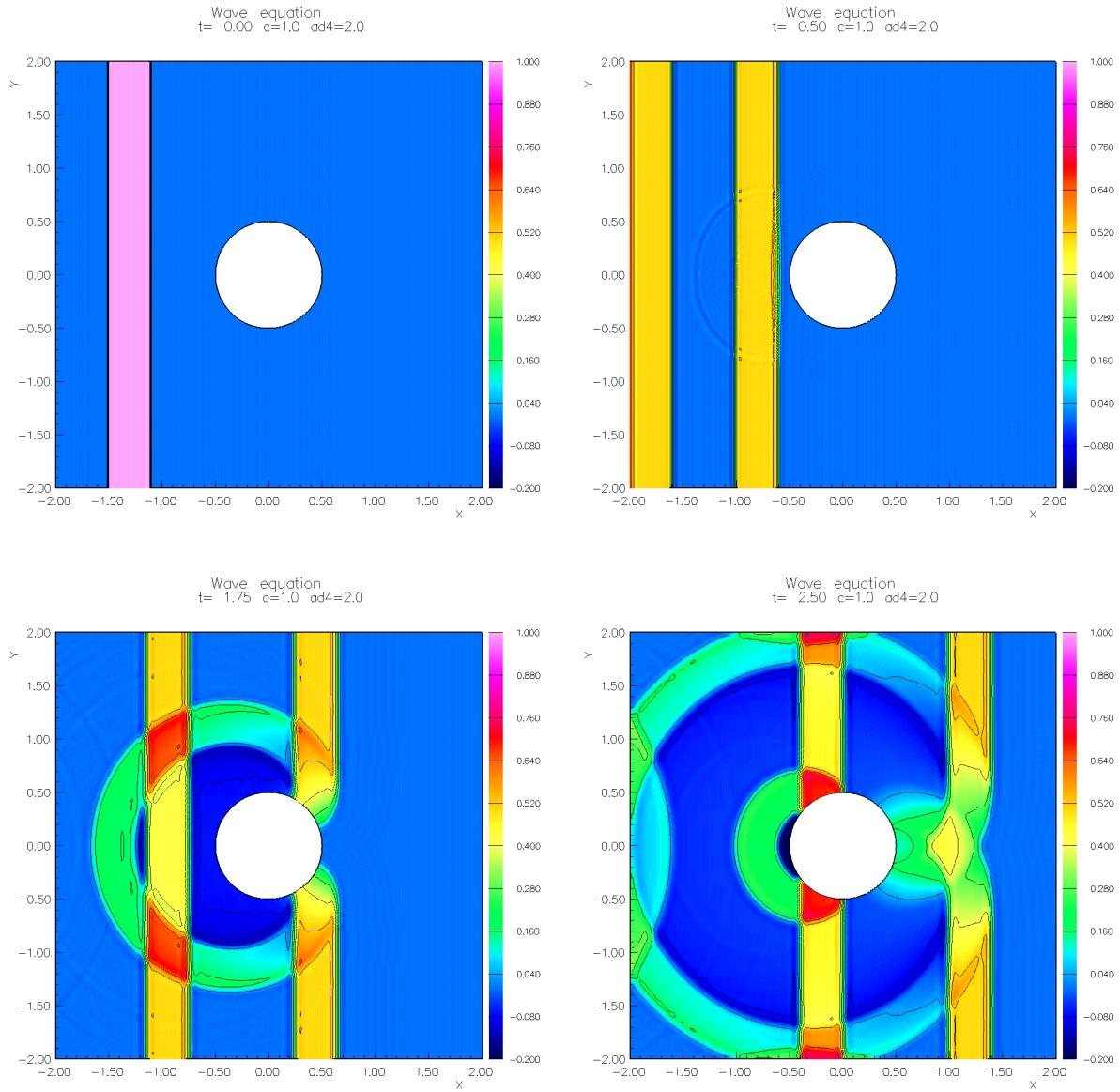


Figure 10: Results from the 2D wave equation. An initial rectangular pulse splits into two and then collides with the cylinder and walls.

3.12 Example: Moving overlapping grids

This example shows how to move a component grid and recompute the overlapping grid. The second component grid will be rotated.

You might try running this example with the grid `sis.hdf` which is an overlapping grid for a square inside a square. When the example runs a window will pop up and a grid will be shown. Choose the menu item `erase` and `exit` (right mouse button) to continue.

The results will also be saved in a show file, called “`move1.show`”. Use Overture/bin/plotStuff to look at this file. (file `Overture/primer/move1.C`)

```

1  #include "Ogen.h"
2  #include "PlotStuff.h"
3  #include "MatrixTransform.h"
4  #include "Ogshow.h"
5
6  //
7  // Moving Grid Example:
8  //   o read in a grid from a data-base file, rotate a component grid and recompute the overlapping grid.
9  //   o interpolate a grid function, update the interpolate for the new grid
10 //   o save solutions in a show file
11 //
12 int
13 main(int argc, char *argv[])
14 {
15     Overture::start(argc,argv); // initialize Overture
16
17     Mapping::debug=0;
18
19     int plotOption=TRUE;
20     if( argc > 1 )
21     { // look at arguments for "noplott"
22         aString line;
23         for( int i=1; i<argc; i++ )
24         {
25             line=argv[i];
26             if( line=="noplott" )
27                 plotOption=False;
28         }
29     }
30
31     aString nameOfOGFile;
32     cout << "Enter the name of the (old) overlapping grid file:" << endl;
33     cin >> nameOfOGFile;
34
35     // Create two CompositeGrid objects, cg[0] and cg[1]
36     CompositeGrid cg[2];
37     getFromADataBase(cg[0],nameOfOGFile);           // read cg[0] from a data-base file
38     //cg[0].update();
39     cg[1]=cg[0];                                // copy cg[0] into cg[1]
40
41     // Rotate component grid 1 (do this by changing the mapping)
42     int gridToMove=1;
43     Mapping & mappingToMove = *(cg[0][gridToMove].mapping().mapPointer);
44
45     // Use this MatrixTransform to change the existing Mapping, the MatrixTransform
46     // can rotate/scale and shift any Mapping, keep a transform for each composite grid
47     MatrixTransform transform0(mappingToMove);
48     MatrixTransform transform1(mappingToMove);
49
50     // Replace the mapping of the component grid that we want to move:
51     cg[0][gridToMove].reference(transform0);
52     cg[0].update();
53     cg[1][gridToMove].reference(transform1);
54     cg[1].update();
55
56     // now we destroy all the data on the new grid -- it will be shared with the old grid
57     // this is not necessary to do but it will save space
58     cg[1].destroy(CompositeGrid::EVERYTHING);
59
60     // we tell the grid generator which grids have changed
61     LogicalArray hasMoved(2);

```

```

62     hasMoved      = LogicalFalse;
63     hasMoved(gridToMove) = LogicalTrue; // Only this grid will move.
64     char buff[80];
65
66     PlotStuff ps(plotOption,"Moving Grid Example");           // for plotting
67     PlotStuffParameters psp;
68     // Here is the overlapping grid generator
69     Ogen gridGenerator(ps);
70
71     enum
72     {
73         rotate,
74         shift
75     } moveOption=rotate;
76
77
78     // Here is an interpolant
79     Interpolant interpolant(cg[0]);
80     realCompositeGridFunction u(cg[0]);
81     // Here is a show file
82     Ogshow show("movel.show");
83     show.setMovingGridProblem(TRUE);
84
85     int numberOfSteps=20;
86     real deltaAngle=5.*Pi/180.;
87     real xShift=-.01;
88
89     // ---- Move the grid a bunch of times.----
90     for (int i=1; i<=numberOfSteps; i++)
91     {
92         int newCG = i % 2;          // new grid
93         int oldCG = (i+1) % 2;      // old grid
94
95         ps.erase();
96         psp.set(GI_TOP_LABEL,sprintf(buff,"Solution at step=%i",i)); // set title
97         PlotIt::plot(ps,cg[oldCG],psp);        // plot the current overlapping grid
98         psp.set(GI_PLOT_THE_OBJECT_AND_EXIT,TRUE);    // set this to run in "movie" mode (after first plot)
99         ps.redraw(TRUE);
100
101        // Rotate the grid by rotating the mapping
102        // After the first step we must double the angle since we start from the old grid
103        if( moveOption==rotate )
104        {
105            real angle = i==1 ? deltaAngle : deltaAngle*2.;
106            if( newCG==0 )
107                transform0.rotate(axis3,angle);
108            else
109                transform1.rotate(axis3,angle);
110        }
111        else
112        {
113            real delta = i==1 ? xShift : xShift*2.;
114            if( newCG==0 )
115                transform0.shift(delta,0.,0.);
116            else
117                transform1.shift(delta,0.,0.);
118        }
119        // Update the overlapping newCG, starting with and sharing data with oldCG.
120        gridGenerator.updateOverlap(cg[newCG], cg[oldCG], hasMoved );
121
122        interpolant.updateToMatchGrid(cg[newCG]);
123        u.updateToMatchGrid(cg[newCG]);
124        // assign values to u
125        Index I1,I2,I3;
126        for( int grid=0; grid<cg[newCG].numberOfComponentGrids(); grid++ )
127        {
128            MappedGrid & mg = cg[newCG][grid];
129            getIndex(mg.dimension(),I1,I2,I3);
130            real freq = 2.*i/numberOfSteps;
131            u[grid](I1,I2,I3)=cos(freq*mg.vertex()(I1,I2,I3,axis1))*sin(freq*mg.vertex()(I1,I2,I3,axis2));
132        }
133        u.interpolate();

```

```
134 // save the result in a show file, every fourth step
135 if( (i % 4) == 1 )
136 {
137     show.startFrame();
138     show.saveComment(0,sprintf(buff,"Solution form move1 ar step = %i",i));
139     show.saveSolution(u);
140 }
141 }
142 cout << "Results saved in move1.show, use Overture/bin/plotStuff to view this file" << endl;
143 Overture::finish();
144 return 0;
145 }
146 }
```

3.13 Example: Adaptive Grids

This example shows how to build an adaptive grid with a refinement level. Starting from a grid collection consisting of a single square, two refinement grids are added. Grid functions are created and the grid functions are plotted on the adaptive grid and the refinement levels.

(file Overture/primer/amrExample1.C)

```

1  #include "Overture.h"
2  #include "SquareMapping.h"
3  #include "AnnulusMapping.h"
4  #include "HDF_DataBase.h"
5  #include "PlotStuff.h"
6  #include "display.h"
7
8  int
9  main(int argc, char *argv[])
10 {
11     Overture::start(argc,argv); // initialize Overture
12
13     printf(" ----- \n");
14     printf(" Demonstrate the method for adding refinement levels to a grid collection \n");
15     printf(" Show how to plot the grid collection, refinement levels, or grid functions \n");
16     printf(" ----- \n");
17
18
19     PlotStuff ps;           // for plotting
20     PlotStuffParameters psp;
21
22     SquareMapping mapping(-1., 1., -1., 1.);           // Create a SquareMapping
23     mapping.setGridDimensions(axis1,11); mapping.setGridDimensions(axis2,11);
24 //AnnulusMapping mapping;           // Create an Annulus
25 //mapping.setGridDimensions(axis1,21); mapping.setGridDimensions(axis2,11);
26
27     MappedGrid mg(mapping);      // grid for a mapping
28     mg.update();
29
30     // Create a two-dimensional GridCollection with one grid.
31     GridCollection gc(2,1);
32     gc[0].reference(mg);
33     gc.updateReferences();
34     gc.update(MappedGrid::THEvertex);
35
36     psp.set(GI_TOP_LABEL,"initial grid"); // set title
37     PlotIt::plot(ps,gc,psp);           // plot the grid
38
39     // assign values to the grid collection function
40     realGridCollectionFunction u(gc);
41     Index I1,I2,I3;
42     int grid;
43     for( grid=0; grid<gc.numberOfComponentGrids(); grid++ )
44     {
45         getIndex(gc[grid].dimension(),I1,I2,I3);
46         u[grid](I1,I2,I3)=sin(gc[grid].vertex()(I1,I2,I3,axis1)*Pi)*sin(gc[grid].vertex()(I1,I2,I3,axis2)*Pi);
47     }
48     ps.erase();
49     psp.set(GI_TOP_LABEL,"u on initial grid");
50     PlotIt::contour(ps,u,psp);
51
52     gc.update(GridCollection::THErefinementLevel); // indicate that we are want a refinement level
53
54     // Add a refinement, specify position in the coarse grid index space
55     IntegerArray range(2,3), factor(3);
56     range(0,0) = 2; range(1,0) = 6;
57     range(0,1) = 2; range(1,1) = 6;
58     range(0,2) = 0; range(1,2) = 0;
59     factor = 4;           // refinement factor = 4
60     Integer level = 1;
61     grid = 0;             // refine this base grid
62     gc.addRefinement(range, factor, level, grid); // add a refinement grid to level 1
63
64     range(0,0) = 4; range(1,0) = 8;
65     range(0,1) = 4; range(1,1) = 8;

```

```

66     range(0,2) = 0; range(1,2) = 0;
67     gc.addRefinement(range, factor, level, grid); // add another refinement grid to level 1
68
69     gc.update(GridCollection::THErefinementLevel);
70
71     gc.refinementFactor.display("gc.refinementFactor");
72     gc.refinementLevel[0].refinementFactor.display("gc.refinementLevel[0].refinementFactor");
73     gc.refinementLevel[1].refinementFactor.display("gc.refinementLevel[1].refinementFactor");
74
75     ps.erase();
76     psp.set(GI_TOP_LABEL,"refined grid");
77     PlotIt::plot(ps,gc,psp); // plot the grid collection including refinements
78
79     ps.erase();
80     psp.set(GI_TOP_LABEL,"refinementLevel[0]");
81     PlotIt::plot(ps,gc.refinementLevel[0],psp); // plot refinement level 0 only
82
83     u.updateToMatchGrid(gc); // tell u that the gridCollection has been changed
84
85     gc.update(MappedGrid::THEvertex);
86
87     gc.setMaskAtRefinements();
88
89     for( grid=0; grid<gc.numberOfComponentGrids(); grid++ )
90     {
91         printf(" grid=%i, level=%i, refinementFactor=%i\n",grid,gc.refinementLevelNumber(grid),
92                gc.refinementFactor(axis1,grid));
93
94
95         getIndex(gc[grid].dimension(),I1,I2,I3);
96         u[grid](I1,I2,I3)=sin(gc[grid].vertex()(I1,I2,I3,axis1)*Pi)*sin(gc[grid].vertex()(I1,I2,I3,axis2)*Pi);
97
98     }
99     ps.erase();
100    psp.set(GI_TOP_LABEL,"u on refined grid");
101    PlotIt::contour(ps,u,psp);
102
103    ps.erase();
104    psp.set(GI_TOP_LABEL,"refinementLevel[1]");
105    PlotIt::plot(ps,gc.refinementLevel[1],psp); // plot refinement level 1 only
106
107    // Now plot the refinementLevel's in the grid functions
108    ps.erase();
109    psp.set(GI_TOP_LABEL,"u.refinementLevel[0]");
110    PlotIt::contour(ps,u.refinementLevel[0],psp); // plot u on refinement level 0
111
112    ps.erase();
113    psp.set(GI_TOP_LABEL,"u.refinementLevel[1]");
114    PlotIt::contour(ps,u.refinementLevel[1],psp); // plot u on refinement level 1
115
116    Overture::finish();
117    return 0;
118 }
```

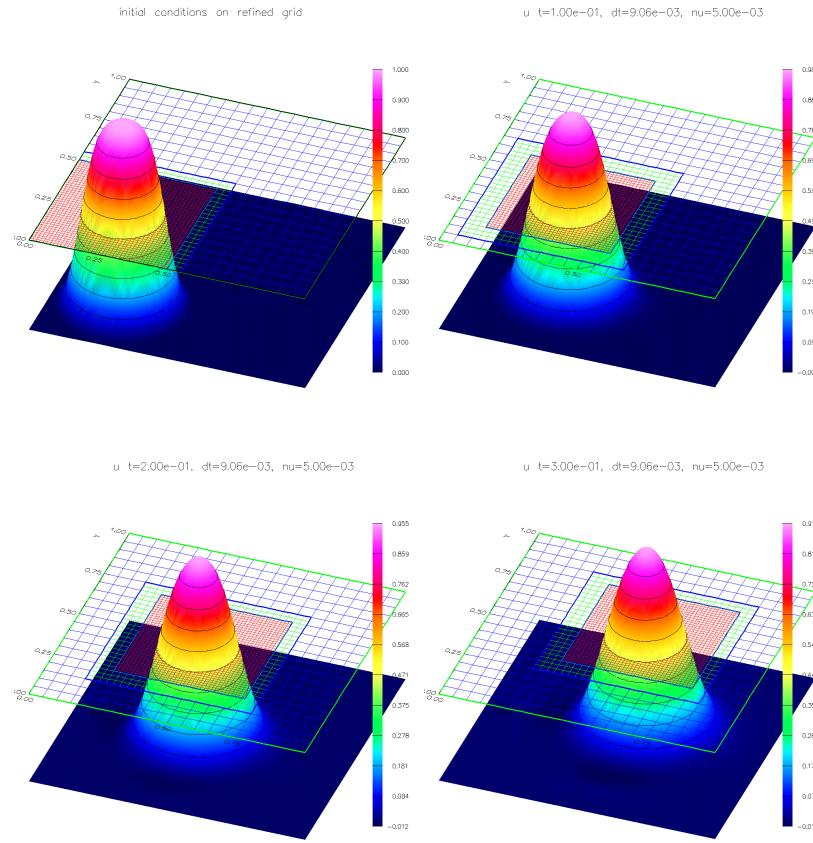


Figure 11: Results from hypeAmr, solving a convection diffusion equation with adaptive mesh refinement.

3.14 Example: Time dependent adaptive mesh refinement solver

The file `Overture/primer/hypeAmr.C` solves a convection diffusion equation using adaptive mesh refinement (AMR). The equation

$$u_t + au_x + bu_y = \nu \Delta u$$

is advanced with a fourth-order accurate Runge-Kutta time stepping algorithm. Every few steps a new AMR grid is computed, based on an estimate of the error. The solution must then be interpolated from the old AMR grid to the new AMR grid. For simplicity, a single time step is used on all grids. The AMR algorithm is implemented with the help of the following classes

ErrorEstimator : class used to compute error estimates.

Regrid : class used to build a new AMR grid.

InterpolateRefinements : used to

- interpolate from one AMR grid to a second AMR grid,
- interpolate ghost-boundaries of refinement grids
- interpolate coarse grid points that are hidden by refinement grids.

This class in turn uses the **Interpolate** class which knows how to interpolate refinement patch points from a coarser grid.

See the documents [2],[?] for further details.

3.15 Example: Multigrid Overlapping Grids

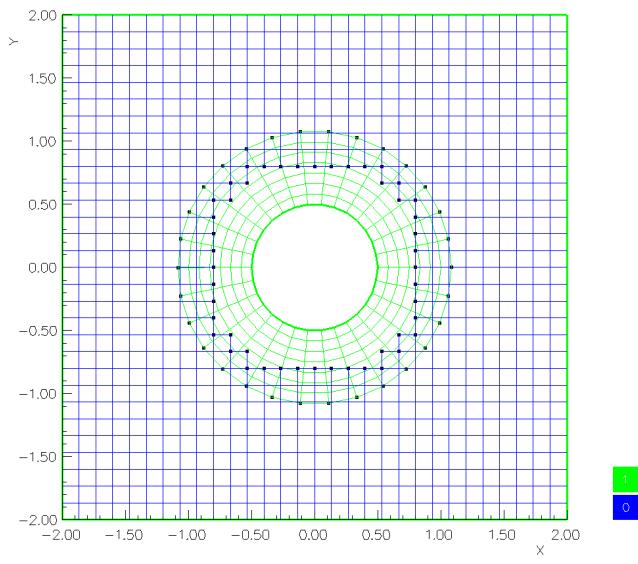
This example shows how to manipulate an overlapping grid that has more than one multigrid level. The overlapping grid should be created with more than one multigrid level as shown for example in the `cicmg.cmd` command file presented in the `ogen` grid generator documentation.

The following program reads in the overlapping grid. It then plots the overlapping grids at the different levels (each multigrid level is a valid `CompositeGrid`). Next a grid function is built and the different multigrid levels of the grid function are assigned and plotted. (file `Overture/primer/mgExample1.C`)

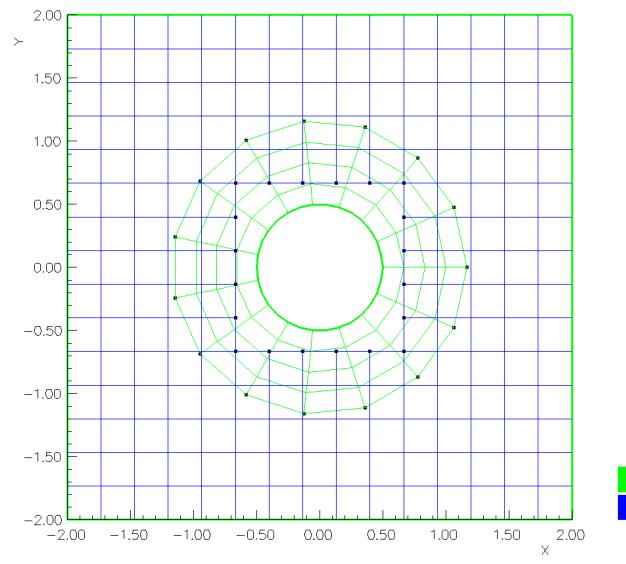
```

1 #include "Overture.h"
2 #include "PlotStuff.h"
3
4 int
5 main(int argc, char* argv[])
6 {
7     ios::sync_with_stdio();
8     Index::setBoundsCheck(On);
9
10    printf(" ----- \n");
11    printf(" Demonstrate how to use multigrid levels with an overlapping grid. \n");
12    printf(" The overlapping grid should be created with more than 1 multigrid level, \n");
13    printf(" see the cicmg.cmd command file as an example. \n");
14    printf(" ----- \n");
15
16    aString nameOfOGFile;
17    cout << "mgExample1>> Enter the name of the (old) overlapping grid file: (cicmg for example)" << endl;
18    cin >> nameOfOGFile;
19
20    // create and read in a CompositeGrid
21    CompositeGrid cg;
22    getFromADataBase(cg,nameOfOGFile);
23    cg.update();
24
25    PlotStuff ps;           // for plotting
26    PlotStuffParameters psp;
27    char buff[80];          // buffer for sPrintf
28
29    // plot each multigrid level (the grid plotter also knows how to plot multigrid levels,
30    // if we were to use: PlotIt::plot(ps,cg); and choose the menu option "plot a multigrid level" )
31    int grid,level;
32    for( level=0; level<cg.numberOfMultigridLevels(); level++ )
33    {
34        psp.set(GI_TOP_LABEL,sprintf(buff,"Multigrid level %i",level)); // set title
35        PlotIt::plot(ps,cg.multigridLevel[level],psp);      // plot the CompositeGrid for a given level
36    }
37
38    // create a grid function on the multigrid-overlapping grid and assign values to it
39    realCompositeGridFunction u(cg);
40    Index I1,I2,I3;
41    for( level=0; level<cg.numberOfMultigridLevels(); level++ )
42    {
43        CompositeGrid & cgLevel = cg.multigridLevel[level];           // make a reference to a given level
44        realCompositeGridFunction & uLevel = u.multigridLevel[level];
45        for( grid=0; grid<cgLevel.numberOfComponentGrids(); grid++ )
46        {
47            getIndex(cgLevel[grid].dimension(),I1,I2,I3);
48            uLevel[grid](I1,I2,I3)=sin(cgLevel[grid].vertex()(I1,I2,I3,0)*Pi) // u = sin(x*Pi)*sin(y*Pi)
49                           *sin(cgLevel[grid].vertex()(I1,I2,I3,1)*Pi);
50        }
51        psp.set(GI_TOP_LABEL,sprintf(buff,"u on multigrid level %i",level)); // set title
52        PlotIt::contour(ps,uLevel,psp);
53    }
54
55    return 0;
56 }
57
58
59

```



(a)



(b)

Figure 12: The two multigrid levels for the grid created by `cicmg.cmd`. A `CompositeGrid` `cg` can hold multiple multigrid levels which are referenced as `cg.multigridLevel[level]`, `level=0,1,...`. Each `cg.multigridLevel[level]` is itself a valid `CompositeGrid`.

3.16 Example: Solving elliptic problems on each multigrid level

This example shows how to solve an elliptic problem on each multigrid level of an overlapping grid. The example also shows how to access the matrix coefficients and compute the residual. The overlapping grid should be created with more than one multigrid level as shown for example in the `cicmg.cmd` command file (see the previous section).

The following program reads in the overlapping grid. Grid functions are built on the multigrid overlapping grid to hold the solution, right-hand-side and the coefficient matrix. Since the `CompositeGridOperators` and the sparse solver class `Oges` do not know about multigrid levels it is necessary to explicitly build these objects for each level. For each multigrid level we build a coefficient matrix and solve a problem. The errors are printed. (file `Overture/primer/mgExample2.C`)

```

1  #include "Overture.h"
2  #include "CompositeGridOperators.h"
3  #include "Oges.h"
4  #include "OGPolyFunction.h"
5
6
7 // These macros define how to access the elements in a coefficient matrix. See the example below
8 #undef C
9 #undef M123
10 #define M123(m1,m2,m3) (m1+halfWidth1+width1*(m2+halfWidth2+width2*(m3+halfWidth3)))
11 #define COEFF(m1,m2,m3,I1,I2,I3) c(M123(m1,m2,m3),I1,I2,I3)
12
13 int
14 main(int argc, char* argv[])
15 {
16     ios::sync_with_stdio();
17     Index::setBoundsCheck(On);
18
19     printf(" ----- \n");
20     printf(" Demonstrate how to solve an elliptic problem on different multigrid levels.\n");
21     printf(" The overlapping grid should be created with more than 1 multigrid level, \n");
22     printf(" see the cicmg.cmd command file as an example. \n");
23     printf(" ----- \n");
24
25     aString nameOfOGFile;
26     cout << "mgExample2>> Enter the name of the (old) overlapping grid file: (cicmg for example)" << endl;
27     cin >> nameOfOGFile;
28
29     // create and read in a (multigrid) CompositeGrid
30     CompositeGrid cgmg;
31     getFromADataBase(cgmg,nameOfOGFile);
32     cgmg.update();
33
34     // allocate operators and sparse solvers for all levels
35     CompositeGridOperators *opMG = new CompositeGridOperators [cgmg.numberOfMultigridLevels()];
36     Oges *solverMG = new Oges [cgmg.numberOfMultigridLevels()];
37     // Now build a coefficient matrix
38     Range all;
39     const int stencilSize=int( pow(3,cgmg.numberOfDimensions())+1.5 );
40     realCompositeGridFunction coeffMG(cgmg,stencilSize,all,all,all);
41
42     // build grid functions for the solution and rhs
43     realCompositeGridFunction uMG(cgmg), fMG(cgmg);
44     // create a twilight-zone function for checking the errors
45     int degreeOfSpacePolynomial = 2;
46     int degreeOfTimePolynomial = 1;
47     int numberOfComponents = cgmg.numberOfDimensions();
48     OGPolyFunction exact(degreeOfSpacePolynomial,cgmg.numberOfDimensions(),numberOfComponents,
49                           degreeOfTimePolynomial);
50
51     // Now loop over each level. Solve a Poisson problem on each level
52     for( int level=0; level<cgmg.numberOfMultigridLevels(); level++ )
53     {
54         // first make some references for ease of use
55         CompositeGrid & cg = cgmg.multigridLevel[level];
56         if( FALSE && level>0 )
57         {
58             cg.interpolateeGrid[0].display("cg.interpolateeGrid");
59             cg.interpolationPoint[0].display("cg.interpolationPoint.display");
60         }
61     }
62

```

```

63     realCompositeGridFunction & coeff = coeffMG.multigridLevel[level];
64     realCompositeGridFunction & u = uMG.multigridLevel[level];
65     realCompositeGridFunction & f = fMG.multigridLevel[level];
66     CompositeGridOperators & op = opMG[level];
67
68     op.updateToMatchGrid(cg); // the operators on this level must be associated with a grid (once only)
69     op.setStencilSize(stencilSize); // set stencil size for operators
70
71     coeff.setIsACoefficientMatrix(TRUE, stencilSize);
72     coeff.setOperators(op);
73     coeff = op.laplacianCoefficients();
74     // fill in the coefficients for the boundary conditions
75     coeff.applyBoundaryConditionCoefficients(0,0,BCTypes::dirichlet, BCTypes::allBoundaries);
76     coeff.applyBoundaryConditionCoefficients(0,0,BCTypes::extrapolate,BCTypes::allBoundaries);
77     coeff.finishBoundaryConditions();
78
79     Oges & solver = solverMG[level];
80     solver.setCoefficientArray( coeff ); // supply coefficients
81     solver.updateToMatchGrid( cg ); // create a solver, and update to the grid (once only)
82
83     // assign the rhs: Laplacian(u)=f, u=exact on the boundary
84     Index I1,I2,I3, Ia1,Ia2,Ia3;
85     int side,axis,grid;
86     Index Ib1,Ib2,Ib3;
87     for( grid=0; grid<cg.numberOfComponentGrids(); grid++ )
88     {
89         MappedGrid & mg = cg[grid];
90         getIndex(mg.indexRange(),I1,I2,I3);
91         if( cg.numberOfDimensions()==1 )
92             f[grid](I1,I2,I3)=exact.xx(mg,I1,I2,I3,0);
93         else if( cg.numberOfDimensions()==2 )
94             f[grid](I1,I2,I3)=exact.xx(mg,I1,I2,I3,0)+exact.yy(mg,I1,I2,I3,0);
95         else
96             f[grid](I1,I2,I3)=exact.xx(mg,I1,I2,I3,0)+exact.yy(mg,I1,I2,I3,0)+exact.zz(mg,I1,I2,I3,0);
97         // loop over the boundaries
98         for( axis=0; axis<mg.numberOfDimensions(); axis++ )
99             for( side=0; side<=1; side++ )
100             {
101                 if( mg.boundaryCondition()(side,axis) > 0 )
102                 {
103                     getBoundaryIndex(mg.gridIndexRange(),side,axis,Ib1,Ib2,Ib3);
104                     f[grid](Ib1,Ib2,Ib3)=exact(mg,Ib1,Ib2,Ib3,0);
105                 }
106             }
107         }
108
109         u=0.; // initial guess for iterative solvers
110         real time0=getCPU();
111         solver.solve( u,f ); // solve the equations
112         real time=getCPU()-time0;
113         cout << "level=" << level << ", time for solve of the Dirichlet problem = " << time << endl;
114
115         real error=0.;
116         for( grid=0; grid<cg.numberOfComponentGrids(); grid++ )
117         {
118             getIndex(cg[grid].indexRange(),I1,I2,I3);
119             where( cg[grid].mask()(I1,I2,I3)!=0 )
120                 error=max(error,max(abs(u[grid](I1,I2,I3)-exact(cg[grid],I1,I2,I3,0))));
121         }
122         printf("level=%i, Maximum error with dirichlet bc's= %e\n",level,error);
123
124         // Now compute the maximum residual
125         real maximumResidual=0.;
126         realCompositeGridFunction residual(cg);
127         // These stencil widths are used by the COEFF macro
128         const int width1=3, halfWidth1=width1/2, width2=3, halfWidth2=width2/2;
129         const int width3= cg.numberOfDimensions()==2 ? 1 : 3, halfWidth3=width3/2;
130
131         for( grid=0; grid<cg.numberOfComponentGrids(); grid++ )
132         {
133             realArray & c = coeff[grid];
134             realArray & uu= u[grid];

```

```

135     realArray & ff= f[grid];
136     realArray & res = residual[grid];
137     // We must first reshape the arrays so that we can multiply by the coefficient matrix
138     uu.reshape(1,uu.dimension(0),uu.dimension(1),uu.dimension(2));
139     ff.reshape(1,ff.dimension(0),ff.dimension(1),ff.dimension(2));
140     res.reshape(1,res.dimension(0),res.dimension(1),res.dimension(2));
141
142     getIndex(cg[grid].indexRange(),I1,I2,I3);
143
144     if( cg.numberOfDimensions()==2 )
145     {
146         // The COEFF macro makes the coeff array look like a 6 dimensional array.
147         res(0,I1,I2,I3)=ff(0,I1,I2,I3)-(
148             COEFF( 0, 0,0,I1,I2,I3)*uu(0,I1 ,I2 ,I3)
149             +COEFF( 1, 0,0,I1,I2,I3)*uu(0,I1+1,I2 ,I3)
150             +COEFF( 0, 1,0,I1,I2,I3)*uu(0,I1 ,I2+1,I3)
151             +COEFF(-1, 0,0,I1,I2,I3)*uu(0,I1-1,I2 ,I3)
152             +COEFF( 0,-1,0,I1,I2,I3)*uu(0,I1 ,I2-1,I3)
153             +COEFF( 1, 1,0,I1,I2,I3)*uu(0,I1+1,I2+1,I3)
154             +COEFF( 1,-1,0,I1,I2,I3)*uu(0,I1+1,I2-1,I3)
155             +COEFF(-1, 1,0,I1,I2,I3)*uu(0,I1-1,I2+1,I3)
156             +COEFF(-1,-1,0,I1,I2,I3)*uu(0,I1-1,I2-1,I3)
157         );
158     }
159     else
160     {
161         res(0,I1,I2,I3)=ff(0,I1,I2,I3)-(
162             COEFF(-1,-1,-1,I1,I2,I3)*uu(0,I1-1,I2-1,I3-1)
163             +COEFF( 0,-1,-1,I1,I2,I3)*uu(0,I1 ,I2-1,I3-1)
164             +COEFF( 1,-1,-1,I1,I2,I3)*uu(0,I1+1,I2-1,I3-1)
165             +COEFF(-1, 0,-1,I1,I2,I3)*uu(0,I1-1,I2 ,I3-1)
166             +COEFF( 0, 0,-1,I1,I2,I3)*uu(0,I1 ,I2 ,I3-1)
167             +COEFF( 1, 0,-1,I1,I2,I3)*uu(0,I1+1,I2 ,I3-1)
168             +COEFF(-1, 1,-1,I1,I2,I3)*uu(0,I1-1,I2+1,I3-1)
169             +COEFF( 0, 1,-1,I1,I2,I3)*uu(0,I1 ,I2+1,I3-1)
170             +COEFF( 1, 1,-1,I1,I2,I3)*uu(0,I1+1,I2+1,I3-1)
171
172             +COEFF(-1,-1, 0,I1,I2,I3)*uu(0,I1-1,I2-1,I3 )
173             +COEFF( 0,-1, 0,I1,I2,I3)*uu(0,I1 ,I2-1,I3 )
174             +COEFF( 1,-1, 0,I1,I2,I3)*uu(0,I1+1,I2-1,I3 )
175             +COEFF(-1, 0, 0,I1,I2,I3)*uu(0,I1-1,I2 ,I3 )
176             +COEFF( 0, 0, 0,I1,I2,I3)*uu(0,I1 ,I2 ,I3 )
177             +COEFF( 1, 0, 0,I1,I2,I3)*uu(0,I1+1,I2 ,I3 )
178             +COEFF(-1, 1, 0,I1,I2,I3)*uu(0,I1-1,I2+1,I3 )
179             +COEFF( 0, 1, 0,I1,I2,I3)*uu(0,I1 ,I2+1,I3 )
180             +COEFF( 1, 1, 0,I1,I2,I3)*uu(0,I1+1,I2+1,I3 )
181
182             +COEFF(-1,-1, 1,I1,I2,I3)*uu(0,I1-1,I2-1,I3+1)
183             +COEFF( 0,-1, 1,I1,I2,I3)*uu(0,I1 ,I2-1,I3+1)
184             +COEFF( 1,-1, 1,I1,I2,I3)*uu(0,I1+1,I2-1,I3+1)
185             +COEFF(-1, 0, 1,I1,I2,I3)*uu(0,I1-1,I2 ,I3+1)
186             +COEFF( 0, 0, 1,I1,I2,I3)*uu(0,I1 ,I2 ,I3+1)
187             +COEFF( 1, 0, 1,I1,I2,I3)*uu(0,I1+1,I2 ,I3+1)
188             +COEFF(-1, 1, 1,I1,I2,I3)*uu(0,I1-1,I2+1,I3+1)
189             +COEFF( 0, 1, 1,I1,I2,I3)*uu(0,I1 ,I2+1,I3+1)
190             +COEFF( 1, 1, 1,I1,I2,I3)*uu(0,I1+1,I2+1,I3+1)
191         );
192     }
193     // reshape the arrays back to their original shape -- this would
194     // be essential if the GridFunctions u,f or res were used again
195     uu.reshape(uu.dimension(1),uu.dimension(2),uu.dimension(3));
196     ff.reshape(ff.dimension(1),ff.dimension(2),ff.dimension(3));
197     res.reshape(res.dimension(1),res.dimension(2),res.dimension(3));
198
199     // compute the residual at all discretization points
200     where( cg[grid].mask()(I1,I2,I3) > 0 )
201         maximumResidual=max(maximumResidual,max(fabs(res(I1,I2,I3))));
```

}

printf("level=%i, Maximum residual with dirichlet bc's= %e\n",level,maximumResidual);

}

return 0;

207 }

For the grid cicmg.hdf the output is

```
ultrabert{henshaw}74: mgExample2
A++ Internal_Index bounds checking: ON
-----
Demonstrate how to solve an elliptic problem on different multigrid levels.
The overlapping grid should be created with more than 1 multigrid level,
see the cicmg.cmd command file as an example.
-----
mgExample2>> Enter the name of the (old) overlapping grid file: (cicmg for example)
.../cgsh/cicmg
getFrom DataBase: number of CompositeGrid(s) found =1, name[0]=cic
>>>>> SparseRep::update to match grid <<<<<<
>>>>> SparseRep::update to match grid <<<<<<
level=0, time for solve of the Dirichlet problem = 0.165335
level=0, Maximum error with dirichlet bc's= 2.058983e-03
level=0, Maximum residual with dirichlet bc's= 4.582405e-04
>>>>> SparseRep::update to match grid <<<<<<
>>>>> SparseRep::update to match grid <<<<<<
level=1, time for solve of the Dirichlet problem = 0.0532484
level=1, Maximum error with dirichlet bc's= 6.177902e-03
level=1, Maximum residual with dirichlet bc's= 1.449585e-04
ultrabert{henshaw}75:
```

3.17 Example: Calling a fortran function for each component grid.

In this example we solve the equation $u_t = f(u, \mathbf{x}, t)$ on an overlapping grid. A Fortran function is called to compute $f(u, \mathbf{x}, t)$ for each component grid (file Overture/primer/callingFortran.C)

```

1  #include "Overture.h"
2  #include "Ogshow.h"
3  #include "CompositeGridOperators.h"
4  #include "PlotStuff.h"
5  #include "display.h"
6
7  // fortran names have an appended underscore:
8  #define mySolver mysolver_
9
10 extern "C"
11 {
12     // fortran variables are passed by reference:
13     void mySolver( const real &t, const real &dt,const real &a,const real &b,const real &n, const int&nd,
14         const int &nd1a,const int &nd1b,const int &nd2a,const int &nd2b,const int &nd3a,const int &nd3b,
15         const int &nla,const int &nlb,const int &n2a,const int &n2b,const int &n3a,const int &n3b,
16             const real &x,const real &u, real &dudt );
17 }
18
19 int
20 main(int argc, char *argv[])
21 {
22     Overture::start(argc,argv); // initialize Overture
23
24     printf(" -----\n");
25     printf(" Solve a PDE u.t=f(u,x,t). Call a fortran routine to compute f(u,x,t) for \n");
26     printf(" each component grid. Plot the results. \n");
27     printf(" ----- \n");
28
29     aString nameOfOGFile;
30     cout << "callingFortran>> Enter the name of the (old) overlapping grid file:" << endl;
31     cin >> nameOfOGFile;
32
33     // create and read in a CompositeGrid
34     CompositeGrid cg;
35     getFromADataBase(cg,nameOfOGFile);
36     cg.update(MappedGrid::THEvertex | MappedGrid::THEmask); // build vertices and mask
37
38     Interpolant interpolant(cg); // Make an interpolant
39
40     CompositeGridOperators operators(cg); // operators for a CompositeGrid
41
42     Range all;
43     realCompositeGridFunction u(cg,all,all,all,1); // create grid functions
44     realCompositeGridFunction dudt(cg,all,all,all,1);
45     u.setOperators(operators);
46     u.setName("u"); // name the grid function
47
48     u=1.; // initial condition
49     dudt=0.;
50
51     real t=0, dt=.01; // initialize time and time step
52     real a=1., b=1., nu=.1; // initialize parameters
53
54     bool openGraphicsWindow=TRUE;
55     PlotStuff ps(openGraphicsWindow,"callingFortran");
56     PlotStuffParameters psp;
57
58
59     aString buff; // buffer for sPrintf
60     int numberOfTypeSteps=200;
61     for( int i=0; i<numberOfTypeSteps; i++ ) // take some time steps
62     {
63         if( ( i % 10)==0 )
64         {
65             psp.set(GI_TOP_LABEL,sPrintf(buff,"Solution t=%f",t)); // set title
66             ps.erase();
67             PlotIt::contour(ps,u,psp);

```

```

68     ps.redraw(true);
69     psp.set(GI_PLOT_THE_OBJECT_AND_EXIT,TRUE);      // set this to run in "movie" mode (after first plot)
70 }
71
72 int grid;
73 for( grid=0; grid<cg.numberOfComponentGrids(); grid++ ) // loop over grids
74 {
75     MappedGrid & mg = cg[grid];
76     realArray & ug = u[grid];
77     realArray & dudtg = dudt[grid];
78     realArray & x = mg.vertex(); // array of vertices
79
80     const IntegerArray & d = mg.dimension();
81     const IntegerArray & gir= mg.gridIndexRange();
82     const int nd=cg.numberOfDimensions();
83
84     // call a fortran function to compute du/dt
85     // (This function does not currently solve the convection diffusion equation)
86     mySolver( t,dt,a,b,nu,nd, d(0,0),d(1,0),d(0,1),d(1,1),d(0,2),d(1,2),
87               gir(0,0),gir(1,0),gir(0,1),gir(1,1),gir(0,2),gir(1,2),
88               *x.getDataPointer(),*ug.getDataPointer(), *dudtg.getDataPointer() );
89
90     // display(ug,"ug","%6.3f");
91     // display(dudtg,"dudtg","%6.3f");
92
93     ug+=dt*dudtg;
94 }
95 t+=dt;
96 u.interpolate();                                // interpolate
97 // u.display("u after interpolate");
98
99 // apply a dirichlet BC on all boundaries:
100 u.applyBoundaryCondition(0,BCTypes::dirichlet,BCTypes::allBoundaries,0.);
101 u.applyBoundaryCondition(0,BCTypes::extrapolate,BCTypes::allBoundaries,0.);
102 u.finishBoundaryConditions();
103 }
104
105 psp.set(GI_PLOT_THE_OBJECT_AND_EXIT,false);
106 psp.set(GI_TOP_LABEL,sprintf(buff,"Solution t=%f",t)); // set title
107 ps.erase();
108 PlotIt::contour(ps,u,psp);
109
110 Overture::finish();
111 return 0;
112
113 }
```

The fortran function is defined in the file `Overture/primer/mySolver.f`

4 Single versus double precision

Overture is designed so that one writes a single code that can be used in either single or double precision. The types `real`, `realMappedGridFunction`, `realCompositeGridFunction` etc. are either `float` or `double` depending on whether the Overture library has been built with the double precision option. Note that the Overture library must be entirely recompiled for double precision. Thus the steps to take to run in **double precision** are

1. Build the Overture library in double precision.
2. remake any overlapping grids with the new double precision version of the grid generator.

Thus one would never explicitly create a `doubleMappedGridFunction` (except in very special cases). The file `Overture/include/OvertureDefine.h` will define the macro `OV_USE_DOUBLE` if Overture has been compiled in double precision.

5 Makefile's and .cshrc files

When Overture is installed using the Overture/configure script Makefile's are build in sub-directories of Overture. You can copy the Makefile from the Overture/primer directory to use as a starting point for making other applications.

The Overture Makefile's require a number of environmental variables to be defined so that it knows where to find various libraries. Here are some example .cshrc files that could be used to define these environmental variables.

NOTE that if you are making (i.e. installing) Overture (as opposed to just using it) you must explicitly define the locations of OpenGL, HDF and APlusPlus (rather than using the relative paths given below).

Note: If you have made the **double precision** version of Overture then you should add -DDOUBLE to the CCFLAGS definitions in the Makefiles below as in CCFLAGS= -DDOUBLE -O ...

5.1 Sun Workstations

Here is a .cshrc file and a Makefile that can be used on a Sun workstation.

File Overture/primer/cshrc.sun:

```

1  #
2  # a sample .cshrc to use with Overture on a sun
3  # NOTE: If you are making Overture (as opposed to just using it) you must explicitly
4  # define the locations of OpenGL, HDF and APlusPlus (rather than using the relative path below)
5  #
6  setenv XLIBS      /usr/openwin
7  setenv MOTIF       /usr/dt
8  setenv Overture   /n/c19s3/Overture/Overture.v12
9  # use full path name for next three variables if you are installing Overture
10 setenv OpenGL     $Overture/OpenGL
11 setenv HDF        $Overture/HDF
12 setenv APlusPlus  $Overture/A++
13 # this next variable tells the run time loader where to find dynamic (.so) libraries
14 setenv LD_LIBRARY_PATH ${MOTIF}/lib:${XLIBS}/lib:${HDF}/lib:$Overture/lib:${APlusPlus}

```

File Overture/primer/Makefile.sun:

```

1  #
2  # Makefile for a sun (to be combined with .cshrc.sun)
3  # assumes the existence of the file mappedGridExample1.C (which can be copied from $Overture/primer)
4  #
5  INCLUDE = -I. -I$(Overture)/include -I$(A++)/include -I$(OpenGL)/include
6  CLIBS= -L$(Overture)/lib -lOverture_static -lOverture_static -L$(A++) -lA++ -lA++_static -lA++
7  GLIBS= -L$(OpenGL)/lib -lMesaaux -lMesatk -lMesaGLU -lMesaGL -lGLw -L$(HDF)/lib -lmf hdf -ldf -ljpeg -lz -L$(MOTIF)/
8  -L$(XLIBS)/lib -lxt -lx11 -lxext -lf77 -lM77 -lv77 -lsunmath -lm
9
10 cc= cc
11 CC=      CC
12 FC=      f77
13 FFLAGS = -g
14 CCFLAGS= -O $(INCLUDE)
15
16 .SUFFIXES:
17 .SUFFIXES:.f .o .C .o .c .o
18 .C.o:; $(CC) $(CCFLAGS) -c $*.C
19 .c.o:; $(cc) $(ccFLAGS) -c $*.c
20 .f.o:; $(FC) $(FFLAGS) -c $*.f
21
22 mappedGridExample1 = mappedGridExample1.o
23 mappedGridExample1: $(mappedGridExample1)
24         $(CC) $(CCFLAGS) -o mappedGridExample1 $(mappedGridExample1) $(CLIBS) $(GLIBS) -lm
25

```

5.2 SGI Workstations

Here is a .cshrc file and a Makefile that can be used on an SGI machine.

File Overture/primer/cshrc.sgi64:

```

1  #
2  # a sample .cshrc to use with Overture on an sgi
3  # NOTE: If you are making Overture (as opposed to just using it) you must explicitly
4  # define the locations of OpenGL, HDF and APlusPlus (rather than using the relative path below)
5  #

```

```

6 setenv XLIBS /usr
7 setenv MOTIF /usr
8 setenv Overture /usr/src/Overture/Overture.v12
9 # use full path name for next three variables if you are installing Overture
10 setenv OpenGL $Overture/OpenGL
11 setenv HDF $Overture/HDF
12 setenv APlusPlus $Overture/A++
13 # this next variable tells the run time loader where to find dynamic (.so) libraries
14 setenv LD_LIBRARY_PATH $HDF/lib:$Overture/lib:${APlusPlus}

```

File Overture/primer/Makefile.sgi64:

```

1 #
2 # Makefile for irix64 (to be combined with .cshrc.sgi64)
3 # assumes the existence of the file mappedGridExample1.C (which can be copied from $Overture/primer)
4 #
5
6 INCLUDE = -I. -I$(Overture)/include -I$(A++)/include
7 LIBAPP= -L$(A++) -lA++ -lA++_static -lA++
8 CLIBS= -L$(Overture)/lib -lOverture_static -lOverture -lOverture_static -L$(A++) -lA++ -lA++_static -lA++
9 GLIBS= -L$(OpenGL)/lib64 -lGLw -lGL -lGLU -lGLw -L$(HDF)/lib -lmfhdf -ldf -ljpeg -lz -L$(MOTIF)/lib64 -lXm -L$(XLIBS)
10
11 cc= cc
12 CC=      CC
13 FC=      f77
14 FFLAGS = -g
15 CCFLAGS= -64 -mips4 -woff 1188,1047,1681,1021,1110 -Wl,-woff,85 $(INCLUDE)
16
17 .SUFFIXES:
18 .SUFFIXES:.f .o .C .o .c .o
19 .C.o:; $(CC) $(CCFLAGS) -c $*.C
20 .c.o:; $(cc) $(ccFLAGS) -c $*.c
21 .f.o:; $(FC) $(FFLAGS) -c $*.f
22
23 mappedGridExample1 = mappedGridExample1.o
24 mappedGridExample1: $(mappedGridExample1)
25     $(CC) $(CCFLAGS) -o mappedGridExample1 mappedGridExample1.o $(CLIBS) $(GLIBS) -lm

```

5.3 Pentium with Linux

File Overture/primer/Makefile.linux:

```

1 #
2 # Makefile for a linux pentium box
3 #
4 INCLUDE = -I. -I$(Overture)/include -I$(A++)/include -I$(OpenGL)/include
5 CLIBS= -L$(Overture)/lib -lOverture_static -lOverture -lOverture_static -L$(A++) -lA++ -lA++_static -lA++
6 GLIBS= -L$(OpenGL)/lib -lMesaaux -lMesatk -lMesaGLU -lMesaGL -lGLw -L$(HDF)/lib -lmfhdf -ldf -ljpeg -lz -L$(MOTIF)/
7     -L$(XLIBS)/lib -lXt -lX11 -lXext -lXpm -L/usr/lib -lf2c -lgcc -lc -lm
8
9 cc= gcc
10 CC=      g++
11 FC=      g77
12 FFLAGS = -g
13 CCFLAGS= -O $(INCLUDE)
14
15 .SUFFIXES:
16 .SUFFIXES:.f .o .C .o .c .o
17 .C.o:; $(CC) $(CCFLAGS) -c $*.C
18 .c.o:; $(cc) $(ccFLAGS) -c $*.c
19 .f.o:; $(FC) $(FFLAGS) -c $*.f
20
21 mappedGridExample1 = mappedGridExample1.o
22 mappedGridExample1: $(mappedGridExample1)
23     $(CC) $(CCFLAGS) -o mappedGridExample1 $(mappedGridExample1) $(CLIBS) $(GLIBS) -lm

```

6 Variables contained in a MappedGrid

Here we give a brief overview of some of the most important items that are contained in a MappedGrid.

Define the Ranges $R1, R2, R3$ to define all points on a component grid:

```
const int Start=0, End=1, axis1=0, axis2=1, axis3=2;
CompositeGrid cg;
MappedGrid & mg = cg[grid];
Range R1(mg.dimension(Start, axis1), mg.dimension(End, axis1));
Range R2(mg.dimension(Start, axis2), mg.dimension(End, axis2));
Range R3(mg.dimension(Start, axis3), mg.dimension(End, axis3));
Range ND(0, cg.numberofDimensions);
```

Recall that we denote the axes of the unit square (or cube) by r_1, r_2 , (and r_3). Some arrays such as the boundaryCondition array, associate values with each side of a grid. The sides of the grid can be denoted by $r_i = 0$ or $r_i = 1$. These arrays are dimensioned as boundaryCondition($0:1, 0:2$) with

$$\text{boundaryCondition(side, axis)} = \text{value for } r_{\text{axis}} = \text{side} , \text{ side} = 0, 1 , \text{ axis} = 0, 1, 2 \quad (1)$$

Some arrays, such as the array of vertex coordinates, come in three flavours, vertex, vertex2D and vertex1D. The first is dimensioned vertex($R1, R2, R3, ND$) and thus looks like an array for a three dimensional grid. When the grid is two-dimensional the Range $R3$ will only have 1 point. This array is useful when writing a code that will work in both 3D and 2D. The array vertex2D($R1, R2, ND$) is only available when the grid is two-dimensional.

- **IntArray boundaryCondition(0:1,0:2)** Boundary condition flags, positive for a real boundary, negative for a periodic boundary and zero for an interpolation boundary.
- **IntArray boundaryDiscretizationWidth(0:2)** Width of the boundary condition stencil.
- **realMappedGridFunction center(R1,R2,R3,ND)** Coordinates of discretization centres.
- **realMappedGridFunction center2D(R1,R2,ND)** Coordinates of discretization centers, for a two-dimensional grid.
- **realMappedGridFunction center1D(R1,ND)** Coordinates of discretization centers, for a one-dimensional grid.
- **realMappedGridFunction centerDerivative(R1,R2,R3,ND,ND)** Derivative of the mapping at the discretization centers.
- **realMappedGridFunction centerDerivative2D(R1,R2,ND,ND)** Derivative at the discretization centers, for a two-dimensional grid.
- **realMappedGridFunction centerDerivative1D(R1,ND,ND)**
- **FloatMappedGridFunction centerJacobian(R1,R2,R3)** Determinant of centerDerivative.
- **IntArray dimension(0:1,0:2)** Dimensions of grid arrays – actual size of the A++ arrays, including ghostpoints.
- **IntArray discretizationWidth(0:2)** Interior discretization stencil width (default=3)
- **IntArray gridIndexRange(0:1,0:2)** Index range of gridpoints, excluding ghost points.
- **realArray gridSpacing(0:2)** Grid spacing in the unit square, equal to 1 over the number of grid cells.
- **IntArray indexRange(0:1,0:2)** Index range of computational points, excluding ghostpoints and excluding periodic grid lines on the “End”.
- **LogicalR isAllCellCentered** Grid is cell-centred in all directions (variable name misspelled for historical reasons, circa 1776)
- **LogicalR isAllVertexCentered** Grid is vertex-centred in all directions
- **LogicalArray isCellCentered(0:2)** Is this grid cell-centred in each direction.
- **IntArray isPeriodic(0:2)** Grid periodicity, equal one if notPeriodic, derivativePeriodic or functionPeriodic.

- **realMappedGridFunction inverseVertexDerivative(R1,R2,R3,ND,ND)** Inverse derivative of the mapping at the vertices. `inverseVertexDerivative(i1,i2,i3,axis,dir)` is the partial derivative of r_{axis} with respect to x_{dir} .
- **realMappedGridFunction inverseVertexDerivative2D(R1,R2,ND,ND)** Inverse derivative at the vertices, for a two-dimensional grid.
- **realMappedGridFunction inverseVertexDerivative1D(R1,ND,ND)** Inverse derivative at the vertices, for a one-dimensional grid.
- **realMappedGridFunction inverseCenterDerivative(R1,R2,R3,ND,ND)** Inverse derivative at the discretization centers.
- **realMappedGridFunction inverseCenterDerivative2D(R1,R2,ND,ND)** Inverse derivative at the discretization centers, for a two-dimensional grid.
- **realMappedGridFunction inverseCenterDerivative1D(R1,ND,ND)** Inverse derivative at the discretization centers, for a one-dimensional grid.
- **IntMappedGridFunction mask(R1,R2,R3)** mask array that indicates which points are used and not used.
- **MappingRC mapping** Grid mapping (MappingRC is a reference counted Mapping which behaves like the Mapping class)
- **FloatArray minimumEdgeLength(0:2)** Minimum grid cell-edge length.
- **FloatArray maximumEdgeLength(0:2)** Maximum grid cell-edge length.
- **IntR numberOfDimensions** Number of space dimensions, an `IntR` is basically an `int` (used for reference counting).
- **IntArray numberOfGhostPoints(0:1,0:2)** number of ghost points on each side.
- **realMappedGridFunction vertex(R1,R2,R3,ND)** Vertex coordinates.
- **realMappedGridFunction vertex2D(R1,R2,ND)** Vertex coordinates, for a two-dimensional grid.
- **realMappedGridFunction vertex1D(R1,ND)** Vertex coordinates, for a one-dimensional grid.
- **FloatArray vertexBoundaryNormal[3][2]** Outward normal vectors at the vertices on each boundary. These arrays are dimensioned so that they lie on their respective boundary:
 - `vertexBoundaryNormal[0][0](R1.getBase():R1.getBase(),R2,R3,ND),`
 - `vertexBoundaryNormal[0][1](R1.getBound():R1.getBound(),R2,R3,ND),`
 - `vertexBoundaryNormal[1][0](R1,R2.getBase():R2.getBase(),R3,ND),`
 - `vertexBoundaryNormal[1][1](R1,R2.getBound():R2.getBound(),R3,ND),`
 - etc.
- **FloatArray centerBoundaryNormal[3][2]** Outward normal vectors at the centers on each boundary.
- **realMappedGridFunction vertexDerivative(R1,R2,R3,ND,ND)** Derivative of the mapping at the vertices, `vertexDerivative(i1,i2,i3,axis,dir)` is the partial derivative of x_{axis} with respect to r_{dir} .
- **realMappedGridFunction vertexDerivative2D(R1,R2,ND,ND)** Derivative of the mapping at the vertices, for a two-dimensional grid.
- **realMappedGridFunction vertexDerivative1D(R1,ND,ND)** Derivative of the mapping at the vertices, for a one-dimensional grid.
- **FloatMappedGridFunction vertexJacobian(R1,R2,R3)** Determinant of vertexDerivative.

One may specify (or change) which arrays are to exist in the `MappedGrid` by calling the `update` function with an integer bit-flag. The values of the bit flag are determined from the following enumerator

```

enum {
    USEmask = USEgenericGrid << 1,
    USEinverseVertexDerivative = USEmask << 1,
    USEinverseCenterDerivative = USEinverseVertexDerivative << 1,
    USEvertex = USEinverseCenterDerivative << 1,
    USEcenter = USEvertex << 1,
    USEvertexDerivative = USEcenter << 1,
    USEcenterDerivative = USEvertexDerivative << 1,
    USEfaceNormal = USEcenterDerivative << 1,
    USEvertexJacobian = USEfaceNormal << 1,
    USEcenterJacobian = USEvertexJacobian << 1,
    USEvertexBoundaryNormal = USEcenterJacobian << 1,
    USEcenterBoundaryNormal = USEvertexBoundaryNormal << 1,
    USEmappedGrid = USEcenterBoundaryNormal // Do not use.
};

```

- **MappedGrid(const String & file, const String & name)** Constructor from database file and name.
- **MappedGrid(Mapping & mapping)** Constructor from a mapping.
- **void updateReferences()** Set references to reference-counted data.
- **void update(const Int what = USEtheUsualSuspects)** Update the grid.

For further details consult the documentation sitting in the chair in Geoff's office.

7 Variables contained in a CompositeGrid

Define the Ranges

```
const int Start=0, End=1, axis1=0, axis2=1, axis3=2;
CompositeGrid cg;
MappedGrid & mg = cg[grid];
Range R1(mg.dimension(Start, axis1), mg.dimension(End, axis1));
Range R2(mg.dimension(Start, axis2), mg.dimension(End, axis2));
Range R3(mg.dimension(Start, axis3), mg.dimension(End, axis3));
Range ND(0, cg.numberOfDimensions());
Range NG(0, cg.numberOfComponentGrids());
Range MG(0, cg.numberOfMultigridLevels());

Range NI(0, cg.numberOfInterpolationPoints(grid));
```

- **IntR numberOfComponentGrids** Number of component grids (MappedGrid's).
- **IntR numberOfRows** Number of space dimensions.
- **IntArray numberOfRowsInterpolationPoints(NG)** The number of interpolation points on each component grid.
- **LogicalR interpolationIsAllExplicit**
- **LogicalArray interpolationIsImplicit(NG,NG)**
- **IntArray interpolationWidth(3,NG,NG)** The width of the interpolation stencil (direction, toGrid, fromGrid).
- **realArray interpolationOverlap(3,NG,NG)** The minimum overlap for interpolation (direction, toGrid, fromGrid).
- **ListOfReferenceCountedObjects<realArray> interpolationCoordinates[NG](NI,ND)** Coordinates of interpolation point on component grid “grid” are `interpolationCoordinates[grid](n, axis)` for $0 \leq n \leq \text{numberOfInterpolationPoints}(\text{grid})$.
- **ListOfReferenceCountedObjects<IntArray> interpoleeGrid[NG](NI)** Index of the “interpolee grid”, i.e. this is the index of the grid from which we interpolate.
- **ListOfReferenceCountedObjects<IntArray> interpoleeLocation[NG](NI,ND)** Location of interpolation stencil on the interpolee grid, this is the index of the lower left corner of the stencil.
- **ListOfReferenceCountedObjects<IntArray> interpolationPoint[NG](NI,ND)** Indices of interpolation point.
- **ListOfReferenceCountedObjects<realArray> interpolationCondition[NG](NI)** Interpolation condition number.
- **IntGridCollectionFunction mask[NG](R1,R2,R3)** Flag array, positive for discretization point, negative for interpolation point, zero for unused point.
- **ListOfReferenceCountedObjects<MappedGrid> grid[NG]** Here is the list of MappedGrid's.

Here are variables related to multigrid levels

- **IntR numberOfMultigridLevels**
- **IntArray coarseToFineWidth(0:2,NG,MG)** Prolongation stencil width
- **IntArray coarseToFineIsImplicit(NG,MG)** Prolongation is always implicit.
- **IntArray fineToCoarseWidth(0:2,NG,MG)** Restriction stencil width
- **IntArray fineToCoarseIsImplicit(NG,MG)** Restriction is always implicit.
- **IntArray fineToCoarseFactor(0:2,NG,MG)** Ratio of this to coarser level
- **ListOfReferenceCountedObjects<CompositeGrid> compositeGrid**

References

- [1] D. L. BROWN, *Overture operator classes for finite volume computations on overlapping grids, user guide*, Tech. Rep. UCRL-MA-133649, Lawrence Livermore National Laboratory, 1998.
- [2] D. L. BROWN AND W. HENSHAW, *Adaptive mesh refinement routines for Overture*, Research Report UCRL-MA-140918, Lawrence Livermore National Laboratory, 2000.
- [3] W. HENSHAW, *A fourth-order accurate method for the incompressible Navier-Stokes equations on overlapping grids*, J. Comp. Phys., 113 (1994), pp. 13–25.
- [4] ———, *Finite difference operators and boundary conditions for Overture, user guide*, Research Report UCRL-MA-132231, Lawrence Livermore National Laboratory, 1998.
- [5] ———, *Grid functions for Overture, user guide*, Research Report UCRL-MA-132231, Lawrence Livermore National Laboratory, 1998.
- [6] ———, *Mappings for Overture, a description of the Mapping class and documentation for many useful Mappings*, Research Report UCRL-MA-132239, Lawrence Livermore National Laboratory, 1998.
- [7] ———, *Ogen: An overlapping grid generator for Overture*, Research Report UCRL-MA-132237, Lawrence Livermore National Laboratory, 1998.
- [8] ———, *Oges user guide, a solver for steady state boundary value problems on overlapping grids*, Research Report UCRL-MA-132234, Lawrence Livermore National Laboratory, 1998.
- [9] ———, *Ogshow: Overlapping grid show file class, saving solutions to be displayed with plotStuff, user guide*, Research Report UCRL-MA-132235, Lawrence Livermore National Laboratory, 1998.
- [10] ———, *Plotstuff: A class for plotting stuff from Overture*, Research Report UCRL-MA-132238, Lawrence Livermore National Laboratory, 1998.
- [11] ———, *A primer for writing PDE codes with Overture*, Research Report UCRL-MA-132231, Lawrence Livermore National Laboratory, 1998.
- [12] ———, *Other stuff for Overture, user guide, version 1.0*, Research Report UCRL-MA-134292, Lawrence Livermore National Laboratory, 1999.
- [13] ———, *OverBlown: A fluid flow solver for overlapping grids, reference guide*, Research Report UCRL-MA-134289, Lawrence Livermore National Laboratory, 1999.
- [14] ———, *OverBlown: A fluid flow solver for overlapping grids, user guide*, Research Report UCRL-MA-134288, Lawrence Livermore National Laboratory, 1999.
- [15] ———, *Time step determination for pdes with applications to programs written with Overture*, Research Report UCRL-MA-134300, Lawrence Livermore National Laboratory, 1999.
- [16] W. HENSHAW, H.-O. KREISS, AND L. REYNA, *A fourth-order accurate difference approximation for the incompressible Navier-Stokes equations*, Comput. Fluids, 23 (1994), pp. 575–593.
- [17] D. QUINLAN, *A++/P++ class libraries*, Research Report LA-UR-95-3273, Los Alamos National Laboratory, 1995.

Index

- adaptive grids
 - example, 52, 54
- boundary conditions
 - explicit application, 10
- coefficient matrix
 - system example, 23
- CompositeGrid
 - display interactively with gridQuery, 24
 - example, 24
- CompositeGridFunction
 - example, 27
- differentiation, 44
 - of a grid function, example, 33
- double precision, compiling for, 63
- efficient computation, example, 47
 - example1, 24
 - example10, 44
 - example2, 27
 - example3, 30
 - example4, 31
 - example5, 33
 - example6, 35
 - example7, 38
 - example8, 40
 - example9, 42
- finite volume operators
 - example, 13
- gridQuery, 24
- interpolation
 - example, 30
- MappedGrid
 - example, 6
- mappedGridExample1, 6
- mappedGridExample2, 8
- mappedGridExample3, 10
- mappedGridExample3CC, 13
- mappedGridExample4, 15
- mappedGridExample5, 16
- mappedGridExample6, 19
- MappedGridFunction
 - example, 6
- MappedOperators
 - example, 6
- Mapping
 - for writing your own, 15
- moving grids
 - example, 49
- multigrid
 - example, 55
- PDE
 - on a CompositeGrid, example, 35
 - Poisson equation, example, 38
 - solve on a MappedGrid, 8
 - solve on a MappedGrid with explicit boundary conditions., 10
 - steady incompressible Navier-Stokes, 23
 - using finite volume operators, 13
 - wave equation, example, 47
- PlotStuff
 - example, 40
- restart file, 42
- show file
 - example, 31
- time step determination
 - example, 19
- twilight zone
 - example, 16
- wave equation, example, 47